UNIVERZA NA PRIMORSKEM
Fakulteta za matematiko, naravoslovje in informacijske tehnologije
Oddelek za informacijske znanosti in tehnologijo

Sistemi 1

Vaje iz zbirnega jezika

Avtor: Domen Šoberl

2021-2022

Kazalo

1 Števila ...

1.1 Pretvarjanje med številskimi sistemi

Pretvorite naslednja desetiška števila najprej v 16-bitno dvojiško, nato pa še v šestnajstiško obliko:

a) 67 b) 49155 c) 65534

Pretvorite naslednja 16-bitna šestnajstiška števila najprej v dvojiško, nato pa še v desetiško obliko:

a) 1000 b) ABCD c) FFFF

1.2 Številski prostori

Koliko različnih števil lahko zapišemo s toliko biti?

a) 8 b) 10 c) 16 d) 20 e) 32

Koliko bitov potrebujemo, če želimo nasloviti:

a) 14 B b) 64 KB c) 1 MB d) 100 MB e) 1 GB

Denimo, da imamo 16-bitni naslovni prostor. Razdelite ta prostor na štiri bloke enakih velikosti. Zapišite prvi in zadnji naslov vsakega od teh blokov. *Šestnajstiška števila bomo zapisovali s predpono* `0x`.

Koliko naslovov je med `0x1000` in `0x2000`, če vključimo tudi začetni in končni naslov?

Shraniti želimo 1000 števil, vsakega na svoj naslov. Pričnemo za naslovom `0x0200`. Kateri naslov pripada zadnjemu številu?

1.3 Računalniška aritmetika

Izračunajte vsoto šestnajstiških števil:

a) A + B b) A781 + 1942 c) A000 + 7000

Denimo, da uporabljamo zgolj 16-bitna števila. Kakšen je tedaj rezultat seštevanja A000 + 7000?

Naj bo v 16-bitnem prostoru števil $x =$ `0xA123`. Katero število v tem prostoru je ekvivalentno $-x$? Torej, za katero število $y = -x$ velja $x + y = 0$?

2 Uvod v zbirni jezik

Task 2.1 Razložite pomen in funkcijo vseh registrov našega simuliranega računalniškega sistema.

Rešitev:

A, B, C, D	Splošnonamenski registri za izvajanje aritmetično logičnih operacij.
IP	*Instruction pointer* - lokacija naslednjega ukaza, ki naj ga procesor izvede.
SP	*Stack pointer* - lokacija vrha sklada.
SR	*Status register* - statusne zastavice.

Statusne zastavice:

S	*Supervisor mode* - ne uporabljamo pri tem predmetu.
M	*Interrupt mask* - prekinitve so omogočene.
C	*Carry* - pri zadnji izvedeni aritmetični operaciji je prišlo do prenosa bita.
Z	*Zero* - Rezultat zadnje izvedene aritmetične operacije je 0.
F	*Fault* - Prišlo je do napake pri izvajanju programa.
H	*Halt* - Izvajanje programa je ustavljeno.

Task 2.2 Prevedite in zaženite naslednji program v zbirnem jeziku:

```
1 MOV A, 0x10
2 ADD A, 10
3 HLT
```

Zapišite in razložite prevedeno strojno kodo. Sledite izvajanju programa korak po korak in razložite njegovo delovanje.

Rešitev: Prevajalnik je sestavil sledečo strojno kodo: `06 00 00 10 14 00 00 0A 00`.
Razlaga:

```
MOV [06] A [00], 0x10 [00 10]
ADD [14] A [00], 10 [00 0A]
HLT [00]
```

Zgornji program premakne šestnajstiško število 10 v 16-bitni register A in mu prišteje decimalno število 10.

Task 2.3 Premaknite $AL \leftarrow 120$ in $BL \leftarrow 180$. Izvedite 8-bitno in 16-bitno seštevanje teh dveh vrednosti. Razložite različna rezultata.

Rešitev:

```
; 8-bit addition          ; 16-bit addition
MOVB AL, 120              MOVB AL, 120
MOVB BL, 180              MOVB BL, 180
ADDB AL, BL               ADD A, B
HLT                       HLT
```

Rezultat 8-bitnega seštevanja je $A = 44$, 16-bitnega seštevanja pa $A = 300$. Pri 8-bitnem seštevaju pride do preliva (angl. *overflow*), zato se nastavi *carry* bit (C).

Task 2.4 Dve 8-bitni vrednosti x in y sta shranjeni v 16-bitnem registru A, tako da $AH = x$ in $AL = y$. Napišite program, ki ti dve vrednosti premakne tako, da $A = x$ in $B = y$.

Rešitev:

```
MOV B, A
SHR A, 8
AND B, 0x00FF
```

Priprava na izpit:

a) 16-bitna registra A in B hranita 8-bitni vrednosti x in y. Napišite program, ki ti dve vrednosti premakne tako, da $AH = x$ in $AL = y$.

b) Napišite takšen program še samo s 16-bitnimi ukazi in operacijo OR.

c) Kaj naredi sledeči program v strojni kodi: `0x10 0x10 0x00 0x41`?

d) Poženite spodnji program v zbirnem jeziku in opazujte spremembe v statusnem registru. Razložite, zakaj se je spremenil vsak od statustnih bitov.

```
MOV A, 0xFF00
ADD A, 0x0100
HLT
```

3 Pomnilniško naslavljanje in spremenljivke

Razmislite o različnih načinih naslavljanja, ki so predstavljeni v dodatku A.3. Zapomnite si:

- Oglati oklepaji `[ ]` pomenijo dostop do pomnilnika.
- Vsak ukaz CPE lahko izvede največ en dostop do pomnilnika.

Task 3.1 Kakšna je razlika med programoma:

```
MOV D, 100
MOV A, D
```

in

```
MOV D, 100
MOV A, [D]
```

Rešitev: Prvi program naloži desetiško vrednost 100 v register A, drugi program pa v register A naloži karkoli se nahaja na naslovu 100 (`0x64`).

Task 3.2 Shranite šestnajstiško vrednost `0xABCD` na pomnilniški naslov `0x0100`. Kateri naslov shrani kateri del te vrednosti? Sedaj shranite še 8-bitno število `0x33` na pomnilniški naslov `0x0102`. Napišite program, ki vrednosti na naslovu `0x0102` prišteje vrednost na naslovu `0x0100`.

Rešitev:

```
MOV [0x0100], 0xABCD ; 16-bit value x
MOVB [0x0102], 0x33  ; 8-bit value y

; Implementation of x = x + y
MOV A, [0x0100]   ; Get the value of variable x.
MOV B, 0          ; We use B to convert y to a 16-bit value.
MOVB BL, [0x0102] ; Move y to the lower byte of B.
ADD A, B          ; x + y -> A as 16-bit addition.
MOV [0x0100], A   ; Store the result x + y to x.
HLT
```

Naslov `0x0100` hrani vrednost `0xAB`, naslov `0x0101` pa vrednost `0xCD`.

Task 3.3 Definirajte 16-bitno spremenljivko x in jo inicializirajte na vrednost $0xABCD$. Nato vrednost te spremenljivke povečajte za 3. Spremenljivko x definirajte na naslednje načine:

a) kot specifičen pomnilniški naslov `0x0100`,
b) kot oznako oz. *labelo* (angl. *label*) pod programsko kodo,
c) kot labelo nad programsko kodo,
d) kot labelo na pomnilniškem naslovu `0x0100`.

Solutions:

a) Kot specifičen pomnilniški naslov `0x0100`.

```
; The programmer decides that value x is stored at memory address 0x0100.
MOV [0x0100], 0xABCD ; Set x = 0xABCD
; The following three lines implement x = x + 3.
MOV A, [0x0100] ; Get the value of x.
ADD A, 3        ; Add 3 to the value.
MOV [0x0100], A ; Store the new value back to x.
HLT
```

b) Kot labela pod programsko kodo.

```
; The programmer does not care about the actual address of variable x,
; let the assembler determine it.
MOV A, [x]    ; Get the value from address x.
ADD A, 3      ; Add 3 to the value.
MOV [x], A    ; Store the new value back to address x.
HLT           ; Must (!) stop before data begins.
x: DW 0xABCD ; Let label x determine the memory address after HLT.
```

c) Kot labela nad programsko kodo.

```
; Let variable x reside before the program code.
JMP main
x: DW 0xABCD    ; define and initialize x = 0xABCD

; let label main determine the address of the MOV opcode below.
main:
    MOV A, [x] ; Get the value from address x.
    ADD A, 3   ; Add 3 to the value.
    MOV [x], A ; Store the new value back to address x.
    HLT
```

d) Kot labela na pomnilniškem naslovu `0x0100`.

```
; Let variable x reside at a specific address in memory.
MOV A, [x]
ADD A, 3
MOV [x], A

; Instruction to the compiler where in memory to compile the code below.
ORG 0x0100
x: DW 0xABCD
```

Task 3.4 Definirajte 16-bitne spremenljivke x, y in z. Napišite program, ki izračuna $z = 3z - (x+y)/2$.

Rešitev:

```
JMP main
x: DW 3 ; define x = 1
y: DW 5 ; define y = 3
z: DW 7 ; define z = 7

main:
    ; z = z - (x + y)
    MOV A, [z] ; A <- z
    MUL 3      ; A <- 3*z
    MOV B, [x] ; B <- x
    ADD B, [y] ; B <- x + y
    SHR B, 1   ; B <- (x + y)/2
    SUB A, B   ; A <- 3*z - (x + y)/2
    MOV [z], A ; z <- 3*z - (x + y)/2
    HLT
```

Priprava na izpit:

a) Definirajte spremenljivki x in y ter jima priredite poljubni vrednosti. Napišite program, ki zamenja vrednosti spremenljivk x in y.

b) Napišite program, ki izračuna $z = 16 \cdot (2x - y)$. Množenje izvedite s pomikanjem bitov.

c) Napišite program, ki izračuna $z = x^2 - y^2$. Pri kvadriranju si pomagajte z ukazom `MUL`.

4 Pogojni stavki

Task 4.1 Definirajte 16-bitni spremenljivki x in y s poljubnima začetnima vrednostima. Odštejte $x - y$. Kako odštevanje vpliva na statusne zastavice, če:

a) $x > y$, b) $x = y$, c) $x < y$.

Rešitev:

```
JMP main:
x: DW 7
y: DW 5

main:
    MOV A, [x]
    SUB A, [y]
    HLT
```

a) $Z = 0, C = 0$, b) $Z = 1, C = 0$, c) $Z = 0, C = 1$.

Task 4.2 Kaj naredi ukaz `CMP` in kakšna je prednost uporabe ukaza `CMP` pred uporabo ukaza `SUB`?

Rešitev: Ukaz `CMP` izvede enako artitmetično operacijo kot ukaz `SUB`, vendar spremeni le statusne zastavice. Po uporabi ukaza `CMP` obdržimo vrednosti x in y.

Task 4.3 Napišite program, ki primerja x in y. Kako primerjava vpliva na statusni zastavici `C` in `Z` ob vsaki možni relaciji primerjave? Kateri pogojni skoki so povezani s katero relacijo?

Rešitev:

```
JMP main:
x: DW 7
y: DW 5

main:
    MOV A, [x]
    CMP A, [y] ; Use CMP instead of SUB.
    HLT
```

Pogoj	Z	C	Skočni ukaz
$x = y$	1	-	JZ, JE
$x \neq y$	0	-	JNZ, JNE
$x > y$	0	0	JA, JNBE
$x >= y$	-	0	JNC, JAE, JNB
$x < y$	-	1	JC, JB, JNAE
$x <= y$	1 -	- 1	JNA, JBE

Task 4.4 V zbirnem jeziku implementirajte naslednji pogojni stavek:

```
if (x < y)
    x = x + 1;
```

Rešitev: V zbirnem jeziku je včasih lažje pogoj negirati ter preskočiti telo pogojnega stavka kadar je negirani pogoj izpolnjen.

```
JMP main:
x: DW 7
y: DW 5

main:
    MOV A, [x]
    CMP A, [y]
    JAE continue    ; if (x >= y) skip the if body
    INC A
    MOV [x], A
continue:
    HLT
```

Task 4.5 V zbirnem jeziku implementirajte naslednji pogojni stavek:

```
if (x < y) z = y;
else z = x;
```

Rešitev:

```
main:
    MOV A, [x]
    MOV B, [y]
    CMP A, B
    JAE else        ; if not (x < y) go to else
    MOV [z], B      ; z = y (if body)
    JMP continue
else:
    MOV [z], A      ; z = x (else body)
continue:
    HLT             ; exit the if statement
```

Task 4.6 V zbirnem jeziku implementirajte naslednji pogojni stavek:

```
if (x >= 100 && x < 200)
    x = x / 2;
```

Rešitev: Zgornji pogoj lahko preoblikujemo takole:

```
if (x < 100 || x >= 200) {
}
else {
    x = x / 2;
}
```

In ga implementiramo takole:

```
main:
    MOV A, [x]
    CMP A, 100 ; if (x < 100) don't do it
    JC out
    CMP A, 200 ; if (x >= 200) don't do it
    JNC out
    SHR A, 1   ; do it
    MOV [x], A ; x = x / 2
out:
    HLT
```

Priprava na izpit:

V zbirnem jeziku implementirajte naslednje pogojne stavke:

a)
```
if (x + 1 > y)
    x = x - 1;
```

b)
```
if (x % 2 == 0)
    x = x / 2;
```

c)
```
if (x == 0 || x == 2 || x > y) {
    x = x - y;
}
else {
    if (x < y)
        x = x + y;
}
```

5 Zanke

Task 5.1 V zbirnem jeziku implementirajte naslednjo *while* zanko:

```
int i = 0;
while (i < 10) {
    i = i + 1;
}
```

Razmislite o možnostih za optimizacijo te zanke.

Rešitev:

```
JMP main
i: DW 0         ; i = 0
main:
    MOV A, [i] ; Get the value of i.
while:
    CMP A, 10   ; Compare i with 10.
    JAE break   ; If above or equal, end the loop
    MOV A, [i] ; Get the value of i.
    INC A       ; i = i + 1
    MOV [i], A ; Store the new value of i.
    JMP while   ; Loop.
break:
    HLT
```

The loop can be optimized by keeping the value of `i` in a register:

```
JMP main
i: DW 0         ; i = 0
main:
    MOV A, [i] ; Get the value of i.
while:
    CMP A, 10   ; Compare i with 10.
    JAE break   ; If above or equal, end the loop
    INC A       ; i = i + 1
    JMP while   ; Loop.
break:
    MOV [i], A ; Store the last value of i.
    HLT
```

Težave bi lahko nastale, če bi si spremenljivko `i` delile različne niti izvajanja. Ker pa naš sistem izvaja le eno nit, nam glede tega ni treba paziti.

Task 5.2 V zbirnem jeziku implementirajte optimizirano zanko *for*, ki sešteje števila od 1 do 10.

Rešitev: Najprej zapišimo rešitev v programskem jeziku C:

```
int sum = 0;
for (int i = 1; i <= 10; i++) {
    sum = sum + i;
}
```

Ker je spremenljivka `i` lokalna (vidna le znotraj zanke), lahko program optimiziramo tako, da `i` hranimo le v registru, vrednost spremenljivke `sum` pa zapišemo v pomnilnik šele, ko se zanka zaključi.

```
1  JMP main
2  sum: DW 0
3  main:
4      MOV A, 0      ; i = 0
5      MOV B, [sum] ; Get the value of sum.
6  for:
7      CMP A, 10     ; Compare i with 10.
8      JA break      ; if i > 10, finish the loop.
9      ADD B, A      ; sum = sum + i
10     INC A         ; i++
11     JMP for
12 break:
13     MOV [sum], B ; Store the value of sum.
14     HLT
```

Task 5.3 V zbirnem jeziku napišite program, ki izračuna fakulteto danega števila n. Predpostavite, da sistem ne podpira ukaza `MUL`.

Rešitev: V programskem jeziku C bi program lahko zapisali takole:

```
int f = 1;
for (int i = 2; i <= n; i++) { // factorial loop
    f = f * i;
}
```

Ker pa množenje ni dovoljeno, uporabimo seštevanje:

```
int f = 1;
for (int i = 2; i <= n; i++) { // factorial loop
    int fi = 0;
    for (int j = i; j > 0; j--) { // multiplication loop
        fi = fi + f;
    }
    f = fi;
}
```

Rešitev nato pretvorimo v zbirni jezik:

```
JMP main
n: DW 5                 ; The input parameter.
f: DW 1                 ; The result: f = n!
main:
    MOV B, [f]          ; We often need this value.
    MOV A, 2            ; int i = 2
factloop:
    CMP A, [n]          ; Has i surpassed n?
    JA endfactloop      ; yes, finish.
    MOV C, 0            ; int fi = 0
    MOV D, 0            ; int j = 0
multloop:
    CMP D, A            ; Compare j and i.
    JAE endmultloop     ; If j reached i, multiplication is done.
    ADD C, B            ; fi = fi + f
    INC D               ; j++
    JMP multloop
endmultloop:
    MOV B, C            ; f = fi
    INC A               ; i++
    JMP factloop
endfactloop:
    MOV [f], B          ; Store the value of f.
    HLT
```

Priprava na izpit:

a) Program, ki izračuna fakulteto števila n poenostavite z uporabo ukaza `MUL`.

b) Napišite program, ki ugotovi, če je neko dano število deljivo s 3. Problem lahko rešite tako, da od danega števila odštevate konstanto 3, dokler to število ni manjše od 3. Če je rezultat enak 0, je dano število deljivo s 3.

c) Napišite program, ki prešteje, koliko bitov registra A je nastavljenih na vrednost 1. Problem lahko rešite tako, da register pomaknete v levo 16-krat in po vsakem pomiku vrednost izpadlega bita preverite v statusni zastavici `C`.

d) Napišite program, ki preveri, koliko pomnilnika je trenutno prostega. Pristop je sledeči — preberite vsak zlog (angl. *byte*) med in vključno naslovoma `0x0000` in `0x0FFF` ter preštejte ničelne zloge po zadnjem neničelnem zlogu.

6 Sklad in funkcije

6.1 Sklad

- Sklad (angl. *stack*) je linearna podatkovna struktura, ki omogoča neposreden dostop le do zadnjega dodanega elementa.
- Sklad, implementiran v glavnem pomnilniku, CPE omogoča klic funkcij, priročen pa je tudi programerju za začasno shranjevanje podatkov.
- Ukazi CPE za delo s skladom so: `PUSH`, `POP`, `PUSHB`, `POPB`.
- Prepričajte se, da vedno pravilno povežete ukaz POP z ukazom PUSH.

Task 6.1 Inicializirajte sklad. Zamenjajte vrednosti registrov `A` in `B` brez uporabe drugih registrov.

Rešitev:

```
MOV A, 1        ; An initial value of A.
MOV B, 2        ; An initial value of B.
MOV SP, 0x0FFF  ; Initialize the stack pointer.

; One way to switch values between A and B
PUSH A          ; Push the value of A onto stack.
MOV A, B        ; Move the value of B to A.
POP B           ; Pop the previous value of A into B.

; Another way to switch values between A and B
PUSH A          ; Push the value of A onto stack.
PUSH B          ; Push the value of B onto stack.
POP A           ; Pop the previous value of B into A.
POP B           ; Pop the previous value of A into B.
HLT
```

Task 6.2 Na sklad potisnite 100 16-bitnih števil in raziščite tisti del pomnilnika, ki hrani ta števila. Koliko takšnih števil lahko shranite na sklad, preden naletite na težave? Kaj se lahko zgodi?

Rešitev:

```
    MOV SP, 0x0FFF  ; Initialize the stack pointer.
    MOV A, 1        ; Start counting.
loop:
    CMP A, 100      ; When counter exceeds 100
    JA exit         ; exit.
    PUSH A          ; Push the current value on stack.
    INC A           ; Increase the counter.
    JMP loop        ; Next iteration.
exit:
    HLT
```

Naš program zasede 18 zlogov, torej nam ostane še 4078 zlogov prostega pomnilnika. To pomeni, da lahko shranimo do 2039 16-bitnih števil, preden nam zmanjka pomnilnika. Če poskusimo shraniti več kot toliko, sklad začne prepisovati programsko kodo. Na modernem računalniškem sistemu bi operacijski sistem to preprečil in sprožil izjemo *stack overflow*.

6.2 Funkcije

- Klic funkcije izvedite z uporaba ukaza `CALL` (nikoli `JMP`).
- Funkcija se mora vedno vrniti z izvedbo ukaza `RET`.
- Parametre funkciji ter vrnjene vrednosti lahko prenašamo preko registrov ali preko sklada. Izogibajte se prenašanju parametrov preko konstantnih pomnilniških naslovov ali globalnih spremenljivk.

Task 6.3 Napišite funkcijo, ki s sklada povleče dve 16-bitni števili ter nazaj na sklad potisne njuno vsoto. Demonstrirajte uporabo te funkcije.

Rešitev:

```
JMP main

add:
    POP D           ; Pop the return address.
    POP B           ; Pop the second summand.
    POP A           ; Pop the first summand.
    ADD A, B        ; Add the summands.
    PUSH A          ; Push the sum.
    PUSH D          ; Push back the returned address.
    RET             ; Return from function.

main:
    MOV SP, 0xFFF   ; Initialize the stack pointer.
    PUSH 5          ; Push the first summand to stack.
    PUSH 3          ; Push the second summand to stack.
    CALL add        ; Call the function add.
    POP A           ; Retreive the returned sum.
    HLT
```

Task 6.4 Napišite funkcijo `max(a, b)`, ki vrne večje od danih dveh celih števil. Funkcija naj parametra a in b prejme preko registrov A in B.

Rešitev:

```
JMP main
z: DW 0             ; int z = 0;

max:
    CMP A, B        ; If A < B then
    JB returnb      ;   return B
```

```
    RET             ; else the result is already in register A.
returnb:
    MOV A, B        ; Move the result to register A,
    RET             ; and return it.

main:
    MOV SP, 0x0FFF  ; Initialize the stack pointer.
    MOV A, 2        ; Set the first argument.
    MOV B, 5        ; Set the second argument.
    CALL max        ; Call z = max(2, 5).
    MOV [z], A      ; Store the returned value.
    HLT
```

Priprava na izpit:

a) Napišite funkcijo `add(x1, x2, x3, ...)`, ki lahko prejme poljubno število števil $x_1, x_2, \ldots, x_n$ ter v registru A vrne njihovo vsoto. Funkcija naj število n prejme preko registra A, števila pa preko sklada.

b) Napišite funkcijo `sign(x)`, ki vrne predznak danega 16-bitnega predznačenega števila x, torej vrne 1, če je število pozitivno, -1, če je negativno in 0, če je 0. Predpostavimo, da je število x predstavljeno v dvojiškem komplementu.

c) Napišite funkcijo `abs(x)`, ki vrne absolutno vrednost danega 16-bitnega predznačenega števila x. Predpostavimo, da je število x predstavljeno v dvojiškem komplementu.

d) Napišite funkcijo `fib(n)`, ki vrne n-to Fibonaccijevo število.

e) Napišite funkcijo `isprime(x)`, ki preveri, če je dano število x praštevilo. Število je praštevilo, če ni deljivo z nobenim številom med 2 in $n - 1$. Deljivost lahko preverite tako, da število najprej delite (`DIV`), nato pa ga množite (`MUL`) nazaj. Napišite še funkcijo `primes(n)`, ki vrne vsa praštevila med 2 in n. Praštevila lahko vrnete preko sklada, število vseh vrnjenih števil pa preko nekega registra.

7 Polja in nizi

7.1 Polja

Polje (angl. *array*) je del pomnilnika, kamor shranimo zaporedje podatkovnih elementov (npr. števil). Spremenljivka, ki predstavlja polje, hrani naslov njegovega prvega (0-tega) elementa. Spremenljivko, ki hrani pomnilniški naslov, imenujemo *kazalec* (ker kaže na pomnilniško lokacijo). V programskem jeziku je polje pravzaprav le kazalec na prvi element zaporedja podatkov. Za dostop do i-tega elementa v polju moramo poznati velikost elementov (polje lahko hrani le elemente istega tipa, kar pomeni, da so vsi elementi enake velikosti), zato da lahko izračunamo njegov pomnilniški naslov:

$$(\text{naslov } i\text{-tega elementa}) = (\text{naslov 0-tega elementa}) + i \cdot (\text{velikost elementa v zlogih}).$$

Velikost polja običajno določimo vnaprej. Lahko pa uporabimo nek del pomnilniškega prostora (npr. pod programom) kot polje nedoločene velikosti. Pri tem smo omejeni le s količino razpoložljivega pomnilnika.

Task 7.1 Definirajte polje 16-bitnih celih števil:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Napišite program, ki vsak element pomnoži z 10.

Rešitev:

```
1  JMP main
2  numbers:
3      DW 1
4      DW 2
5      DW 3
6      DW 4
7      DW 5
8      DW 6
9      DW 7
10     DW 8
11     DW 9
12     DW 10
13
14 main:
15     MOV C, 0          ; For i from 0 to 9.
16 loop:
17     CMP C, 10         ; If i >= 10,
18     JAE finish        ; finish.
19     MOV D, numbers    ; Compute the address numbers[i].
20     MOV A, C          ; Take index i.
21     MUL 2             ; Each index takes 2 bytes in memory.
```

```
    ADD D, A          ; Add the index to numbers.
    MOV A, [D]        ; Get numbers[i].
    MUL 10            ; Multiply it by 10.
    MOV [D], A        ; Store the result back.
    INC C             ; i++
    JMP loop
finish:
    HLT
```

Task 7.2 Definirajte polje 16-bitnih spremenljivk, ki hrani 1000 elementov ter ga inicializirajte s sodimi števili od 2 do 2000.

Rešitev:

```
JMP main

; The array starts at 0x0100 and ends at 0x08D0 (2000 bytes).
ORG 0x0100
numbers:
ORG 0x08D0

main:
    MOV A, numbers ; The address of the 0-th array position.
    MOV C, 0       ; Array index.
    MOV B, 2       ; Even numbers to be stored in the array.

loop:
    CMP C, 1000    ; If array index >= 1000, break.
    JAE break
    MOV [A], B     ; Store the number to the i-th array position.
    INC C          ; Increase the array index.
    ADD A, 2       ; Compute the address of the next array position.
    ADD B, 2       ; Compute the next even number.
    JMP loop
break:
    HLT
```

7.2 Nizi

Niz (angl. *string*) je polje znakov (8-bitnih vrednosti ASCII). Dolžine nizov se med izvajanjem programa pogosto spreminjajo, zato je potrebna informacija o tem, kateri znak v nizu je zadnji. V zbirnem jeziku se odločimo sami, kako bomo to informacijo hranili. V tem učbeniku bomo uporabili metodo *terminalne ničle*, ki je uporabljena v programskem jeziku C - vsakemu nizu je na konec dodana 8-bitna ničla. Primer:

- `"Hello world"= ['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', 0]`
- `  = [0]` (prazen niz)

Če bi takšnemu nizu pozabili na koncu dodati ničlo, bi program lahko poskušal brati znake preko meje polja ter ostale podatke in programsko kodo v pomnilniku interpretirati kot del tega niza.

Task 7.3 Definirajte niz `s = "Hello world!"`. Napišite funkcijo `length(s)`, ki vrne velikost danega niza. Niz `s` podamo *po referenci* (kot kazalec na prvi znak).

Rešitev:

```
JMP main
s:   DB "Hello world" ; char *s = "Hello world"
     DB 0             ; Terminal value.
len: DW 0             ; Store the length of the string.

; Function length(s) returns the length of the given string s.
; s - string (pointer to the first character) given through register A.
; A 16-bit integer is returned through register A.
length:
    MOV C, A          ; The pointer to the first character.
    MOV A, 0          ; Character counter.
loop:
    MOVB BL, [C]      ; Get the current character.
    CMPB BL, 0        ; Is it a terminal value?
    JE return         ; If yes, we are done.
    INC A             ; Otherwise increase the character counter.
    INC C             ; Compute the address of the next character.
    JMP loop          ; Repeat for the next character.
return:
    RET

main:
    MOV SP, 0x0FFF    ; Initialize the stack.
    MOV A, s          ; Prepare the parameter.
    CALL length       ; Call the function length(s).
    MOV [len], A      ; Store the returned value.
    HLT
```

7.3 Izpisovanje besedila

Na našem sistemu lahko nize izpisujemo na besedilni prikazovalnik tako, da niz preprosto prekopiramo na pomnilniške naslove `0x1000` – `0x101F`. To pomeni, da je besedilni prikazovalnik *pomnilniško preslikan* na te specifične naslove.

Task 7.4 Napišite funkcijo `print(s)`, ki izpiše dani niz.

Rešitev:

```
JMP main
s:   DB "Hello world" ; char *s = "Hello world"
```

```
3       DB 0                ; Terminal value.
4
5  ; Function print(s) prints the given string s to the text display.
6  ; s - string (pointer to the first character) given through register A.
7  print:
8      MOV D, display   ; The pointer to the first display cell.
9  loop:
10     MOVB BL, [A]     ; Get the current character.
11     CMPB BL, 0       ; Is it a terminal value?
12     JE return        ; If yes, we are done.
13     MOVB [D], BL     ; Otherwise copy the character to the display cell.
14     INC A            ; Compute the address of the next character.
15     INC D            ; Compute the address of the next display cell.
16     JMP loop         ; Repeat for the next character.
17 return:
18     RET
19
20 main:
21     MOV SP, 0x0FFF   ; Initialize the stack.
22     MOV A, s         ; Prepare the parameter.
23     CALL print       ; Call the function print(s).
24     HLT
25
26 ; Associate the label 'display' with the memory address 0x01000.
27 ORG 0x1000
28 display:
```

Priprava na izpit:

a) Napišite funkcijo `sum_arrays(a1, a2, len)`, ki prejme (po referenci) polji `a1` in `a2` 16-bitnih celih števil, obe polji sta velikosti `len`, in po elementih sešteje $\texttt{a1} \leftarrow \texttt{a1} + \texttt{a2}$. Funkcija ne vrača vrednosti.

b) Napišite funkcijo `str_empty(s)`, ki prejme niz in vrne vrednost 1, če je niz prazen ter vrednost 0 sicer.

c) Napišite funkcijo `str_cmp(s1, s2)`, ki prejme dva niza poljubnih dolžin in vrne vrednost 1, če sta niza enaka ter vrednost 0 sicer.

d) Napišite funkcijo `count_words(s)`, ki prejme niz in vrne število besed v tem nizu. Besede so ločene z enim ali več presledki.

e) Napišite funkcijo `print(s, line)`, ki izpiše dani niz v prvo (`line` $= 0$) ali drugo (`line` $= 1$) vrstico besedilnega prikazovalnika.

f) Do sedaj smo polja (in nize) funkcijam vedno podajali *po referenci*. Če je funkcija takšno polje spreminjala, so bile spremembe vidne globalno. Razmislite o tem, kako bi polje podali *po vrednosti*. To pomeni, da mora biti kopija celotnega polja, ki ga podajamo funkciji, prenešena preko sklada. Napišite funkcijo `print(s)` tako, da je parameter `s` podan po vrednosti.

8 V/I naprave in prekinitve

Oglejte si vhodno / izhodne registre, ki so našteti v dodatku A.6. Pomnite:

- Ti 16-bitni registri pripadajo različnim V/I napravam ter V/I modulu. Uporabljamo jih za upravljanje in komunikacijo z V/I napravami, ki niso preslikane pomnilniško.
- V/I registre naslavljamo z njihovimi indeksi. Ti indeksi so ločeni od indeksov registrov CPE.
- Do V/I registrov lahko dostopamo le preko CPE registra A, in sicer z uporabo ukazov `IN` in `OUT`.

8.1 Uporaba tipkovnice preko metode *polling*

Najenostavnejši način branja vhodnih podatkov je z uporabo metode *polling* (povpraševanje). Uporabo te metode bomo demonstrirali na primeru branja s tipkovnice. Komunikacija s tipkovnico poteka preko dveh V/I registrov:

- **KBDSTATUS** (indeks 5) - Hrani trenutno stanje tipkovnice v naslednjem formatu:

0	0	0	0	0	0	0	0	0	0	0	0	0	E	U	D

  Ta register je samo za branje (angl. *readonly*).

  - **D = 1** pomeni, da je prišlo do dogodka *keydown*, torej da uporabnik drži pritisnjeno tipko. Koda ASCII trenutno pritisnjene tipke se nahaja v registru KBDDATA.
  - **U = 1** pomeni, da je prišlo do dogodka *keyup*, torej da je uporabnik spustil tipko. Koda ASCII izpuščene tipke se nahaja v registru KBDDATA.
  - **E = 1** pomeni, da je prišlo do napake *buffer overflow*. Zgodil se je nov dogodek tipkovnice, preden je naš program obdelal prejšnjega. To pomeni, da je bil register KBDDATA prepisan z novimi podatki, ASCII koda prejšnje tipke pa je izgubljena. Poskrbimo, da bo naš program vedno obdelal dogodke hitreje, kot jih uporabnik lahko generira s pritiskanjem tipk in tako preprečimo, da bi prišlo do te napake.

- **KBDDATA** (indeks 6) - Hrani kodi ASCII zadnje aktivne tipke. Ta register je samo za branje. Uporabljenih je le spodnjih osem bitov registra, ostali biti so konstanto 0. Ko z ukazom `IN` preberemo vsebino tega registra, se register `KBDSTATUS` samodejno ponastavi na 0.

Task 8.1 Napišite program, ki na besedilnem prikazovalniku izpiše simbol ASCII trenutno pritisnjene tipke. Če trenutno ni pritisnjena nobena tipka, je prikazovalnik prazen. Uporabite metodo *polling*.

Rešitev:

```
readkey:
    IN 5                ; Read the keyboard status.
    CMP A, 0            ; Has anything happened?
```

```
    JE readkey          ; If not, read the keyboard status again.
    MOV B, A            ; Let the status be in register B.
    IN 6                ; Read the key code.
    AND B, 1            ; Mask out the keydown bit.
    CMP B, 1            ; Is this bit set to 1?
    JE display          ; If yes, display the new key code.
    MOVB [0x1000], 0    ; For all other events clear the display.
    JMP readkey         ; And wait for the next keyboard event.
display:
    MOVB [0x1000], AL   ; Write the key code to text display.
    JMP readkey         ; And wait for the next keyboard event.
    HLT                 ; We will never terminate.
```

Priprava na izpit:

a) Napišite program, ki vam omogoča, da na besedilni prikazovalnik natipkate poljubno besedilo dolžine do 16 znakov. Program naj se konča, ko je zaslon poln. Bodite pozorni, da boste dodajali nove znake le ob dogodku *keydown*.

b) Napišite program, ki lahko zazna hkratni pritisk dveh tipk. Denimo, da uporabljamo štiri tipke: '8' - za gor, '2' - za dol, '4' - za levo in '6' - za desno. Ko držimo eno od teh tipk, naj se na prikazovalniku izpiše "GOR", "DOL", "LEVO" ali "DESNO". Če ni pritisnjena nobena tipka, naj bo prikazovalnik prazen. Nato omogočite pritisk dveh tipk hkrati, kar pomeni gibanje po diagonali. Če uporabnik pritisne '8' in '4', naj se izpiše "LEVO GOR". Na ta način bi uporabnik lahko določil osem smeri premikanja. To naredimo tako, da se odzivamo na oba dogodka tipkovnice, *keydown* in *keyup*, pri tam pa prištevamo (*keydown*) in odštevamo (*keyup*) smeri. Določene kombinacije tipk, npr. '8' in '2', se tako medsebojno izničijo.

8.2 Prekinitve

Metoda *polling* je enostavna za uporabo, toda porabi veliko procesorskega časa, zato se jo v praksi izogibamo. Bolj učinkovit pristop je takšen, da V/I naprave CPE same obvestijo, da je nek vhodni podatek pripravljen za branje. Le tedaj CPE prebere statusne in podatkovne registre. Takšno *obvestilo* imenujemo *zahteva za prekinitev* (angl. *interrupt request*, IRQ), saj takšna zahteva CPE prekine pri izvajanju programa, zato da se posveti vhodni napravi. Ko pride do zahteve za prekinitev, CPE takoj skoči na naslov `0x0003` (ta naslov imenujemo *prekinitveni vektor*[1] oz. po angleško *interrupt vector*) in začne na tem naslovu izvajati programsko kodo, ki se tam nahaja. Zato na ta naslov umestimo poseben del programske kode, ki se imenuje *prekinitvena rutina* oz. angleško *Interrupt Service Routine* (ISR). Naloga te rutine je ugotoviti, do kakšne prekinitve je prišlo, postreči napravo, ki je prekinitev zahtevala ter umakniti obdelano zahtevo za prekinitev. CPE se nato vrne točno na tisto mesto, kjer je bila prekinjena.

[1] Prekinitveni vektorji se od procesorja do procesorja razlikujejo, moderni procesorji pa imajo več kot enega.

Zahteve za prekinitve urejamo preko treh V/I registrov:

- **IRQMASK** (indeks 0) - Ta register programer uporabi zato, da omogoči (1) ali onemogoči (0) zahteve za prekinitve posameznih naprav. Format je naslednji:

0	0	0	0	0	0	0	0	0	0	0	0	0	G	T	K

  - **K** - Zahteve tipkovnice za prekinitev.
  - **T** - Zahteve časovnika za prekinitev.
  - **G** - Zahteve grafične kartice za prekinitev.

  Privzeto so vse zahteve za prekinitev onemogočene.

- **IRQSTATUS** (indeks 1) - Sporoča, katere naprave trenutno čakajo na prekinitev. Format tega registra je enak kot format registra IRQMASK. Ta register je samo za branje.

- **IRQEOI** - Preko tega registra umaknemo izbrano zahtevo za prekinitev (EOI pomeni *End Of Interrupt*). Takoj, ko v registru IRQEOI postavimo nek bit, se istoležni bit v registru IRQSTATUS počisti. Ker je register IRQSTATUS samo za branje, bitov v njem ne moremo spreminjati neposredno.

Neodvisno od nastavitve v registru IRQMASK, prekinitve omogočimo ali onemogočimo globalno preko zastavice M statusnega registra. Privzeto so prekinitve onemogočene ($M = 0$). Prekinitve programsko omogočimo z ukazom `STI` (*set interrupts*) in onemogočimo z ukazom `CLI` (*clear interrupts*).

Prekinitev ne moremo gnezditi[2]. To pomeni, da CPU med izvajanjem rutine ISR ne more biti prekinjen. Vsaka zahteva za prekinitev, ki se zgodi med izvajanjem ISR, je zabeležena v register IRQSTATUS. Takoj, ko se CPE vrne iz rutine ISR, ponovno skoči nazaj v ISR, da bi obdelal zahteve ostalih čakajočih naprav.

8.2.1 Zahteva za prekinitev

Ko naprava sproži zahtevo za prekinitev, se izvede naslednje zaporedje akcij:

1. Tisti bit v registru IRQSTATUS, ki ustreza tej napravi, se postavi na 1.
2. Če so prekinitve omogočene globalno ($M = 1$) in so v registru IRQMASK omogočene prekinitve te naprave, potem:
   (a) Če je procesor ustavljen ($H = 1$), se zastavica *halt* počisti ($H = 0$) in procesor se ponovno aktivira.
   (b) Statustni register in povratni naslov (IP) se v tem vrstnem redu potisneta na sklad.
   (c) Prekinitve se onemogočijo globalno ($M = 0$). Na ta način se prepreči gnezdenje prekinitev.
   (d) Procesor skoči na naslov `0x0003` (prekinitveni vektor).

[2]Kompleksnejši sistemi omogočajo gnezdenje prekinitev glede na prioriteto. To pomeni, da zahteva za prekinitev z višjo prioriteto lahko prekine prekinitveno rutino z nižjo prioriteto.

8.2.2 Vračanje iz prekinitvene rutine

Ukaz IRET (*return from interrupt*) izvede naslednje:

1. Povratni naslov (IP) in statusni register se v tem vrstnem redu restavrirata s sklada.
2. Skok na povratni naslov.

S tem, ko se s sklada restavrira statusni register, se restavrira tudi zastavica $M = 1$, kar takoj spet omogoči prekinitve. Če v tem trenutku register IRQSTATUS ni počiščen (nekatere naprave še vedno čakajo na prekinitev), je prekinitvena rutina nemudoma spet poklicana.

8.2.3 Kako napisati prekinitveno rutino

- Najelegantnejši način za umestitev rutine ISR v program je dodati ukaz `JMP isr` v drugo vrstico, takoj za ukaz `JMP main`. Na ta način določimo dve vstopni točki — *zagonsko* in *prekinitveno* vstopno točko.
- Prekinitvena rutina naj bo čim krajša in čim hitrejša. Izogibajte se zankam. Prekinitvena rutina naj zgolj prebere vhodne podatke in nastavi vrednosti globalnim spremenljivkam. Za vse ostalo naj poskrbi glavni program.
- Prepričajte se, da prekinitvena rutina pred vrnitvijo umakne zahtevo po prekinitvi, ki jo je pravkar obdelala, sicer se bo ista zahteva nemudoma spet sprožila.
- Prekinitvena rutina naj obdela le eno prekinitev, nato pa se vrne. Če na prekinitev čaka še kakšna naprava, bo ta postrežena ob naslednjem klicu prekinitvene rutine.

Priporočena struktura programa v zbirnem jeziku, ki uporablja prekinitve:

```
JMP main
JMP isr

; Variables that are being shared between the main program and the ISR.

isr:
    ; Push registers to stack.
irq1:
    ; Check if IRQ 1. If not, jump to IRQ 2.
    ; Process IRQ 1.
    ; Set EOI for IRQ 1.
    JMP iret
irq2:
    ; Check if IRQ 2. If not, jump to IRQ 3.
    ; Process IRQ 2.
    ; Set EOI for IRQ 2.
    JMP iret
    ; ...
iret:
    ; Pop registers from stack.
```

```
    IRET

; Global variables and functions.

main:
    ; Initialize stack.
    ; Mask interrupts.
    STI
    ; Do something or HLT.
```

8.2.4 Prekinitve tipkovnice

Task 8.2 Napišite program, ki na prikazovalnik izpiše simbol ASCII trenutno pritisnjene tipke, toda sedaj uporabite prekinitve tipkovnice.

Rešitev:

```
JMP main
JMP isr

isr:
    PUSH A              ; The ISR will use register A.
    IN 5                ; Get the keyboard status.
    AND A, 1            ; Mask out the keypress event.
    CMP A, 1            ; Is it the keypress event?
    JE printkey         ; If yes, print out the key code.
    IN 6                ; Get the key code to clear the keyboard status.
    MOVB [0x1000], 0    ; Clear the display.
    JMP keysdone
printkey:
    IN 6                ; Get the key code.
    MOVB [0x1000], AL   ; Print out the key code.
keysdone:
    MOV A, 1            ; Keyboard interrupt mask.
    OUT 2               ; Keyboard has been serviced.
iret:
    POP A               ; Restore the register A.
    IRET                ; Return from ISR.

main:
    MOV SP, 0x0FFF      ; Initialize the stack pointer.
    MOV A, 1            ; Keyboard interrupt mask.
    OUT 0               ; Set the mask.
    STI                 ; Enable interrupts.
    HLT                 ; Do nothing.
```

Pri zgornji rešitvi bodite pozorni na naslednje:

- Prekinitvena rutina ni preverjala registra IRQSTATUS, saj so bile prekinitve tipkovnice edine omogočene in zato edine možne.
- V primeru dogodka *keyup* smo prebrali kodo tipke (vrstica 10), čeprav je nismo uporabili. To je bilo potrebno zato, da smo počistili register KBDSTATUS, ki mora biti počiščen, če želimo umakniti zahtevo tipkovnice za prekinitev (vrstica 18).

8.2.5 Prekinitve časovnika

Naš sistem ima vgrajen strojni časovnik, kar programu omogoča, da meri pretečeni čas. Časovnik sicer lahko prebiramo z metodo *polling*, toda njegova resnična uporabnost je v možnosti proženja prekinitev v rednih intervalih. S časovnikom upravljamo preko dveh V/I registrov:

- **TMRPRELOAD** (indeks 3) - Začetna vrednost časovnika. Z nastavljanjem te vrednosti določamo število urninih period med dvema zaporednima zahtevama za prekinitev.
- **TMRCOUNTER** (indeks 4) - Trenutna vrednost časovnika. Ta vrednost se zmanjša vsako urino periodo. Ko doseže 0, se proži zahteva za prekinitev, njegova vrednost pa je ponastavljena na vrednost v registru TMRPRELOAD.

Task 8.3 Napišite program, ki šteje sekunde od 0 do 9 in se nato ustavi. Nastavite frekvenco CPE na 5 Khz in uporabite prekinitve časovnika.

Rešitev:

```
1  JMP main
2  JMP isr
3
4  quit: DW 0              ; quit = false
5
6  isr:
7      PUSH A              ; The ISR will use register A.
8      MOVB AL, [0x1000]   ; Get the currently displayed character.
9      INCB AL             ; Increase its ASCII value.
10     MOVB [0x1000], AL   ; Print out the new value.
11     CMPB AL, '9'        ; Check if '9' has been reached.
12     JNE timerdone       ; If not, we are done.
13     MOV [quit], 1       ; Otherwise set quit to true.
14 timerdone:
15     MOV A, 2            ; Timer interrupt mask.
16     OUT 2               ; Timer has been serviced.
17     POP A               ; Restore the register A.
18     IRET                ; Return from ISR.
19
20 main:
21     MOV SP, 0x0FFF      ; Initialize the stack pointer.
22     MOVB [0x1000], '0'  ; Display '0' on display.
23     MOV A, 5000         ; 1 second = 5000 clock cycles.
24     OUT 3               ; Preset the timer to 5000.
```

```
25      MOV A, 2            ; Timer interrupt mask.
26      OUT 0               ; Set the mask.
27      STI                 ; Enable interrupts.
28  loop:
29      MOV A, [quit]       ; Read the global quit variable.
30      CMP A, 1            ; Check it's value.
31      JE break            ; If true, break the loop.
32      JMP loop            ; Otherwise, read the value again.
33  break:
34      CLI                 ; Disable interrupts.
35      HLT                 ; Terminate.
```

Priprava na izpit:

a) Animirajte prikaz napredka (angl. *progress bar*), ki se prične s prazno vrstico na besedilnem prikazovalniku, nato pa vsake 0.25 sekunde doda simbol '*', dokler prva vrstica prikazovalnika ni polna. Program naj se nato ustavi.

b) Animacijo iz prejšnje točke dopolnite tako, da se napredek ob dogodku *keydown* ponastavi na začetek.

c) Napišite program, ki v prvi vrstici prikazovalnika animira drseče besedilo, dolgo vsaj 20 znakov. Animacija naj bo ciklična.

d) Napišite program, ki prikazuje simbole ASCII vseh trenutno pritisnjenih tipk. Možen pristop je takšen: Definirajte polje (angl. *array*) 256 logičnih vrednosti (true/false), ki označujejo trenutno pritisnjene tipke. Dogodka *keydown* in *keyup* nato preklapljata ustrezne indekse med vrednostima *true* in *false*. Posebna funkcija *print* nazadnje prekopira indekse z vrednostjo *true* na prikazovalnik, kot simbole ASCII.

e) Na sredini prve vrstice besedilnega prikazovalnika prikažite simbol '*'. Če ni pritisnjena nobena tipka, naj simbol miruje. Če uporabnik drži pritisnjeno tipko '4' oz. '6', naj se zvezdica premika levo oz. desno, s hitrostjo 0.25 sekunde na celico. Če uporabnik hkrati drži tipki '4' in '6', naj zvezdica miruje, saj se obe smeri medsebojno izključujeta. Ostale tipke nimajo nobene funkcije.

9 Grafika

V razdelku A si oglejte umestitev grafične kartice v arhitekturo našega sistema. Pomnite:

- CPE z grafično kartico komunicira preko V/I modula.
- V/I registri, ki pripadajo grafični kartici, so VIDMODE, VIDADDR in VIDDATA.
- Grafična kartica uporablja 64 KB lastnega video pomnilnika (VRAM).

Kot je opisano v razdelku A.7, grafična kartica video pomnilnik lahko uporablja na dva različna načina; to sta besedilni način (angl. *text mode*) in rastrski način (angl. *bitmap mode*). Rastrski način je lažje razumljiv in bolj intuitiven, zato si ga bomo ogledali najprej.

9.1 Točkovni način

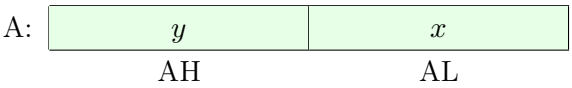
Točkovni (ali rastrski) način omogočimo tako, da nastavimo vrednost registra VIDMODE na 2. Opazili boste, da se takoj po preklopu v ta način na sredini zaslona prikažejo na videz naključni vzorci pik. Razlog za to je v dejstvu, da grafični pomnilnik VRAM na začetku ni prazen. Točkovni način preprosto prikazuje vsebino celotnega pomnilnika na zaslonu, in sicer tako, da se vsak zlog ujema z eno piko (angl. *pixel*). Prvi zlog (`0x0000`) VRAM-a je prikazan kot zgornja leva pika; naslovi se nato večajo od leve proti desni ter proti dnu zaslona, tako da je zadnji zlog (`0xFFFF`) VRAM-a prikazan kot spodnja desna pika. Vsebino VRAM-a lahko v celoti pobrišemo tako, da register VIDMODE nastavimo na vrednost 3. Zaslon je tako v trenutku prazen, vrednost registra VIDMODE pa se povrne v nazadnje uporabljen grafični način, v našem primeru torej na vrednost 2.

9.1.1 Naslavljanje posameznih pik

Če želimo spremeniti barvo neke pike, moramo najprej izračunati njen naslov v VRAM-u. Če je ločljivost zaslona enaka $width \times height$, je naslov pike (x, y) očitno enak

$$address = y \cdot width + x.$$

Ker pa je ločljivost našega zaslona 256×256 pik, je izračun naslova pike v našem primeru še preprostejši. Spomnite se, da množenje števila y s konstanto 256 (širina našega zaslona) dosežemo z 8-kratnim bitnim pomikom števila y v levo. Če torej v registru A računamo 16-bitni naslov pike (x, y), pri čemer sta x in y 8-bitni števili, naslov preprosto nastavimo takole:



Do podatkov v VRAM-u dostopamo preko V/I registrov VIDADDR in VIDDATA. Takoj, ko v VIDADDR zapišemo nek 16-bitni naslov, se 8-bitni podatek s tega naslova pojavi v spodnjem delu registra VIDDATA, medtem ko je njegov zgornji del enak 0. Ta podatek lahko nato spremenimo na poljubno 8-bitno vrednost, tako da prepišemo vrednost registra VIDDATA. Pri tem se le spodnji del registra dejansko zapiše v VRAM.

9.1.2 Barvna paleta

V našem točkovnem načinu je vsaka zaslonska pika predstavljena z enim zlogom, kar pomeni 8-bitno barvno globino oz. 256 možnih barv za vsako piko. Naša grafična kartica uporablja standardni barvni format 3-3-2, kjer so trije najbolj levi biti uporabljeni za zapis intenzivnosti rdeče, srednji trije biti za zapis intenzivnosti zelene, desna dva bita pa za zapis intenzivnosti modre barvne komponente[1]. Očitno je, da vrednost 0 predstavlja črno, vrednost 255 pa belo barvo. Toda kako določimo vse ostale barve?

Za primer si oglejmo barvo z indeksom 252. To število najprej pretvorimo v 8-bitno dvojiško obliko in tako dobimo:

1	1	1	1	1	1	0	0
rdeča			zelena			modra	

To pomeni, da je barva 252 mešanica najintenzivnejšega rdečega in zelenega odtenka, kar nam da najintenzivnejši odtenek rumene barve.

Slika 9.1 prikazuje celotno barvno paleto 3-3-2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	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

Slika 9.1: Barva paletta 3-3-2.

[1] Razlog, da je bil en bit odvzet modri komponenti, ne pa rdeči ali zeleni, je v tem, da je človeško oko manj občutljivo na modre barvne odtenke.

9.1.3 Točkovno risanje v zbirnem jeziku

Task 9.1 Na sredini črnega zaslona prikažite piko modrozelene (angl. *cyan*) barve (mešanica modrega in zelenega odtenka).

Rešitev:

Piko prikažimo na lokaciji $x = 128$, $y = 128$. Indeks modrozelene barve je 31.

```
MOV A, 2       ; Set the bitmap mode
OUT 7          ; through the VIDMODE register.
MOV A, 3       ; Clear the video memory
OUT 7          ; through the VIDMODE register.
MOV A, 0x7F7F  ; Set the address of the pixel
OUT 8          ; through the VIDADDR register.
MOV A, 31      ; Set the cyan color
OUT 9          ; through the VIDATA register.
HLT
```

Task 9.2 Izrisujte pike naključnih barv na naključnih lokacijah, dokler uporabnik ne pritisne neke tipke. Naključna 16-bitna števila lahko generirate tako, da berete V/I register RNDGEN.

Rešitev:

```
    MOV A, 2       ; Set the bitmap mode.
    OUT 7
    MOV A, 3       ; Clear the video memory.
    OUT 7
loop:
    IN 10          ; Get a random number
    OUT 8          ; to set a random location.
    IN 10          ; Get a random number
    OUT 9          ; to set a random color.
    IN 5           ; Read the key status.
    CMP A, 0       ; If there is an event,
    JNE exit       ; exit the loop.
    JMP loop
exit:
    HLT
```

Task 9.3 Na črnem zaslonu narišite dve diagonalni črti, ena naj bo rdeče, druga pa rumene barve. Končna slika bo tako v podobi simbola X.

Rešitev:

```
    MOV A, 2       ; Set the bitmap mode.
    OUT 7
    MOV A, 3       ; Clear the video memory.
    OUT 7
```

```
    MOV C, 0       ; Left diagonal pixel.
    MOV D, 255     ; Right diagonal pixel.
loop:
    MOV A, C       ; Set the address of the left pixel.
    OUT 8
    MOV A, 224     ; Turn the pixel red.
    OUT 9
    MOV A, D       ; Set the address of the right pixel.
    OUT 8
    MOV A, 252     ; Turn the pixel yellow.
    OUT 9
    CMP C, 0xFFFF  ; Is the left diagonal finished?
    JE exit
    ADD C, 257     ; Left diagonal progresses down right.
    ADD D, 255     ; Right diagonal progresses down left.
    JMP loop
exit:
    HLT
```

Priprava na izpit:

a) Napišite funkcijo `set_pixel(x, y, color)`, ki preko registrov AL in AH prejme 8-bitni vrednosti `x` in `y`, preko registra CL pa 8-bitno vrednost `color` ter na lokaciji (x, y) na zaslonu izriše piko v barvi `color`.

b) Napišite funkcijo `get_pixel(x, y)`, ki v registru CL vrne trenutno barvo pike na podani lokaciji (x, y).

c) Napišite funkcijo `rectangle(x0, y0, x1, y1, color)`, ki preko registra A prejme lokacijo (x_0, y_0) zgornjega levega kota, preko registra B lokacijo (x_1, y_1) spodnjega desnega kota, preko registra CL barvo `color` ter izriše zapolnjen pravokotnik s podanimi lastnostmi.

d) Napišite program, ki v prvo vrstico zaslona izriše celotno paleto 3-3-2. Program nato razširite tako, da se paleta ponovi v vsaki vrstici.

e) Napišite funkcijo `convert_color(red, green, blue)`, ki preko sklada prejme 8-bitne barvne komponente RGB in v registru CL vrne najbližji barvni približek znotraj palete 3-3-2. *Namig: Upoštevajte le najpomembnejše bite podanih barvnih komponent RGB.*

9.2 Besedilni način

Verjetno ste opazili, da je točkovna grafika počasna in da zahteva veliko grafičnega pomnilnika. To ni problematično za moderne računalnike, toda za enostavne sisteme, kot je naš[2], je točkovna grafika praktično neuporabna, če želimo izdelovati grafične animacije ali igre. Verjetno si tudi lahko predstavljate, kako težko bi bilo v točkovnem načinu na zaslon izpisati neko besedilo. Zato naša grafična kartica podpira

[2] Večina računalnikov iz obdobja 8-bitnih sistemov se je spopadala z istimi problemi.

tudi *besedilni način*, ki nudi bogat nabor funkcionalnosti, ki CPE razbremenijo kompleksnega računanja položajev pik in njihovih barv[3].

Razlike med *besedilnim* in *točkovnim* načinom so pri naši grafični kartici naslednje (glejte tudi razdelek A.7.1):

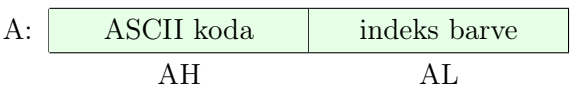
- V besedilnem načinu osnovni grafični element ni pika, ampak znak oz. črka velikosti 16×16. Privzeto so to znaki ASCII, lahko pa jih preoblikujemo v poljubne oblike.
- Besedilno polje, ki vsebuje grafično vsebino, je veliko večji od vidnega zaslona. Zaslon je le okno, ki ga lahko nad besedilnim poljem svobodno premikamo in tako prikaže poljuben del besedilne vsebine.
- Barvna paleta je privzeto 3-3-2, lahko pa jo preoblikujemo v poljuben 8-bitni nabor RGB barv.
- Na zaslon lahko postavimo do osem znakov, ki jih po zaslonu premikamo neodvisno od preostale vsebine. To je znano kot *sprite grafika*, ki je bila na 8-bitnih sistemih tipično uporabljena pri igrah, kjer se *agenti* svobodno premikajo.

Razlog, da je besedilni način hitrejši in zahteva manj grafičnega pomnilnika kot točkovni način je, da mora CPE za zapolnitev celotnega zaslona namesto 65536 pik nanj postaviti le 256 elementov.

Besedilni način omogočimo tako, da vrednost V/I registra VIDMODE postavimo na 1. V tem načinu je uporaba grafičnega pomnilnika VRAM bistveno bolj strukturirana kot v točkovnem načinu. Kot je prikazano v razdelku A.7.1, je za določanje grafične vsebine uporabljena le zgornja polovica VRAM-a. Ta del pomnilnika je na začetku prazen. Spodnja polovica VRAM-a hrani oblike vseh 256 ASCII znakov ter definicijo barvne palete 3-3-2. Ker so te informacije shranjene v VRAM-u (ne pa zakodirane v strojni logiki)[4], jih lahko v svojem programu kadarkoli poljubno spreminjamo, spremembe pa pridejo do izraza nemudoma. V spodnjem delu VRAM-a so shranjene tudi informacije o trenutni barvi ozadja zaslona, položaju zaslonskega okna nad večjim besedilnim poljem ter o osmih prosto premikajočih se znakih (sprite-ih). Preostalega dela VRAM-a grafična kartica v besedilnem načinu ne uporablja, program pa ga po potrebi lahko uporablja kot dodatno shrambo.

9.2.1 Izpisovanje besedila na zaslon

V besedilnem načinu se grafična kartica vede približno podobno kot naš pomnilniško preslikan besedilni prikazovalnik, ki ga že znamo uporabljati. Namesto v RAM, znake sedaj zapisujemo v VRAM, na voljo pa imamo veliko več prostora, poleg tega pa so znaki lahko različnih barv. Namesto 8 bitov na znak sedaj uporabljamo 16 bitov — zgornji del za ASCII kodo, spodnji del pa za indeks barve:



Ko torej na besedilnem zaslonu naslavljate individualne celice, bodite pozorni, da se vedno premikate naprej in nazaj po dva zloga.

[3]Sodobne grafične kartice prevzemajo podobno vlogo, vendar večinoma v zvezi s prikazovanje 3D grafike.

[4]To je tudi razlog, da v točkovnem načinu zaslon začetno ni prazen, saj vizualizira definicije znakov in barv, ki so shranjene v VRAM-u. To je privzeta vsebina VRAM-a, ko grafično kartico vključimo ali programsko ponastavimo (reset), kar storimo tako, da v register VIDMODE zapišemo vrednost 4.

Task 9.4 Na zaslon v rumeni barvi izpišite niz “Hello world!”.

Rešitev:

```
JMP main

s: DB "Hello world!"
   DB 0

main:
    MOV A, 1      ; Set up the text mode
    OUT 7         ; through register VIDMODE.
    MOV C, s      ; Pointer to string.
    MOV D, 0      ; Pointer to screen cells.
loop:
    MOV A, D      ; Activate the screen cell
    OUT 8         ; through register VIDADDR.
    MOVB AH, [C]  ; Get a character.
    CMPB AH, 0    ; The end of the string?
    JE exit       ; If yes, terminate.
    MOVB AL, 252  ; Set it's color to yellow.
    OUT 9         ; Display the character.
    INC C         ; Next character.
    ADD D, 2      ; Next screen cell.
    JMP loop
exit:
    HLT
```

Task 9.5 Izpišite celotno tabelo ASCII tako, da so vsi znaki vidni na zaslonu.

Rešitev: Glavni izziv te naloge je ugotoviti, kdaj moramo preiti v novo vrstico. Besedilno polje je veliko 128×128 znakov, medtem ko je vidni del velik le 16×16 znakov. Če znake izpišemo preko 16. stolpca, ti na zaslonu ne bodo vidni.

Indeksi stolpcev in vrstic so v naslovu VRAM-a razvidni na naslednji način:

Naslov VRAM:	0	indeks vrstice	zlog stolpca
		7 bits	8 bits

Vodilna ničla sovpada z dejstvom, da je zgornji del VRAM-a rezerviran za besedilno polje. Sledi 128 vrstic (7 bitov) in 256 zlogov na vrstico (8 bitov). Očitno je, da če zlog stolpca delimo z 2, dobimo indeks stolpca. Zadostuje torej, če preberemo spodnji del naslova VRAM in preverimo, če smo že preko 32. tloga (16. stolpca). V tem primeru preskočimo preostalih 112 nevidnih celic (224 zlogov), desno od roba vidnega zaslona.

```
    MOV A, 1      ; Set up the text mode
    OUT 7         ; through register VIDMODE.
    MOVB CL, 0    ; ASCII code.
```

```
    MOV D, 0       ; Pointer to screen cells.
loop:
    MOV A, D       ; Activate the screen cell
    OUT 8          ; through register VIDADDR.
    MOVB AH, CL    ; Set the character.
    MOVB AL, 255   ; Set it's color to white.
    OUT 9          ; Display the character.
    INCB CL        ; Next character.
    CMPB CL, 0     ; Have we cycled through the whole ASCII?
    JE break       ; If yes, break the loop.
    ADD D, 2       ; Next screen cell.
    MOV B, D       ; Examine the cell address.
    AND B, 0x00FF  ; Mask out the column bytes.
    CMP B, 32      ; Check if past the 16-th column.
    JB loop        ; If not, continue.
    ADD D, 224     ; Otherwise skip the rest of the line.
    JMP loop       ; Repeat for the next character.
break:
    HLT
```

Priprava na izpit:

a) Razširite naš program "Hello world!" na naslednji način. Definirajte dodatno polje (angl. *array*) z imenom *colors*, ki hrani 12 barvnih indeksov. Nato izpišite niz "Hello world!" tako, da bo vsak znak takšne barve, kot določa istoležni indeks v polju *colors*.

b) Napišite funkcijo `print(s, color)`, ki izpiše dani niz `s` v dani barvi `color`.

c) Napišite funkcijo `print_rainbow(s)`, ki izpiše dani niz `s` tako, da je vsaka črka pobarvana z naključno barvo.

d) Izpišite transponirano obliko matrike ASCII, tako da so znaki natisnjeni od zgoraj navzdol, prehod v nov stolpec pa se zgodi, ko je dosežen spodnji rob zaslona.

9.2.2 Spreminjanje barve ozadja in položaja zaslona

Task 9.6 Na modrem ozadju izpišite rumeno besedilo, ki je dovolj dolgo, da se razteza preko desnega roba vidnega zaslona. Nato premaknite zaslon tako, da bo viden zadnji del izpisanega niza.

Rešitev:

```
JMP main

s: DB "A string long enough to go beyond the right border."
   DB 0

main:
    MOV A, 1       ; Set up the text mode
```

```
    OUT 7          ; through register VIDMODE.
    MOV A, 0xA300  ; Background color information
    OUT 8          ; in VRAM.
    MOV A, 3       ; Blue color index.
    OUT 9          ; Make the background blue.
    MOV C, s       ; Pointer to string.
    MOV D, 0       ; Pointer to screen cells.
loop:
    MOV A, D       ; Activate the screen cell
    OUT 8          ; through register VIDADDR.
    MOVB AH, [C]   ; Get a character.
    CMPB AH, 0     ; The end of the string?
    JE break       ; If yes, break the loop.
    MOVB AL, 252   ; Set it's color to yellow.
    OUT 9          ; Display the character.
    INC C          ; Next character.
    ADD D, 2       ; Next screen cell.
    JMP loop
break:
    MOV A, 0xA302  ; Horizontal scroll information
    OUT 8          ; in VRAM.
    MOV A, 650     ; Scroll horizontally by 650 pixels
    OUT 9          ; by overwriting the existing value.
    HLT
```

Task 9.7 Napišite program, ki izpiše znake ASCII od 'A' to 'Z', vsakega v svojo vrstico. Če so pred izpisom posameznega črke zapolnjene že vse vidne vrstice, pomaknite zaslon navzdol za eno vrstico in tako naredite vidni prostor za naslednjo črko.

Rešitev:

```
JMP main

main:
    MOV A, 1        ; Set up the text mode
    OUT 7           ; through register VIDMODE.
    MOVB CL, 'A'    ; Start with the letter 'A'.
    MOV D, 0        ; Pointer to the screen cells.
loop:
    CMPB CL, 'Z'    ; Are we past the letter 'Z'?
    JA break        ; If yes, finish.
    CMPB DH, 16     ; Have we already filled up the first 15 lines?
    JB noscroll     ; If not, no need to scroll.
    MOV A, 0xA304   ; Get the vertical scroll information
    OUT 8           ; from VRAM
    IN 9            ; to register A.
    ADD A, 16       ; Increase it by 16 pixels
    OUT 9           ; and write it back to VRAM.
```

```
noscroll:
    MOV A, D        ; Activate the current screen cell
    OUT 8           ; in VRAM.
    MOVB AH, CL     ; Set up the current ASCII code.
    MOVB AL, 255    ; Make it white.
    OUT 9           ; Display the character.
    INC C           ; Next character
    ADD D, 256      ; Next line.
    JMP loop
break:
    HLT
```

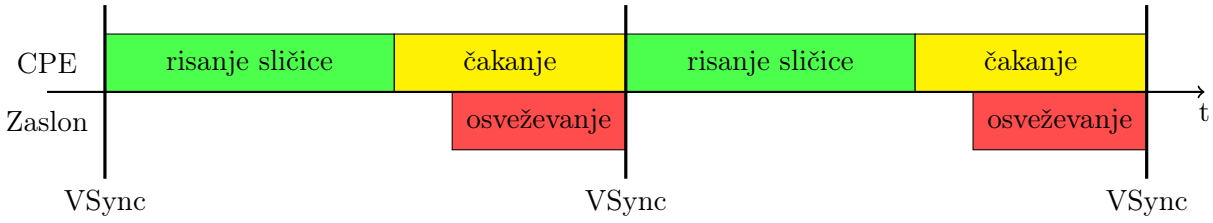
Priprava na izpit:

a) Program, ki izpiše dolgo besedilo in se premakne desno na konec niza spremenite tako, da bo besedilo izpisano vertikalno, program pa bo okno premaknil *navzdol* do konca besedila. To lahko dosežete tako, da v obstoječi rešitvi spremenite le dve vrednosti.

b) Napišite funkcijo `writeln(s)`, ki dani niz izpiše v novo vrstico, pri tem pa ohrani obstoječe besedilo, ki je bilo izpisano s predhodnimi klici te funkcije. Če je na novo izpisan niz pod spodnjim vidnim robom, se zaslon samodejno pomakne navzdol za eno vrstico in izpisani niz tako naredi viden.

9.3 Animacija

Ko se ukvarjamo z računalniško animacijo, je čas pomemben faktor, ki ga moramo pri tem upoštevati. Vsak računalniški zaslon ima neko lastno frekvenco osveževanja, ki določa, kolikokrat na sekundo se izriše celotna slika. Računalniška animacija je zaporedje sličic — v angleški terminologiji se takšna sličica imenuje *frame* — ki morajo biti sinhronizirane s frekvenco osveževanja zaslona. Če niso sinhronizirane, lahko prihaja do neželenih vizualnih učinkov *trganja* (angl. *tearing*) ali utripanja (angl. *flickering*), kar je posledica delnega prikazovanja sličic oz. prikazovanja nedokončanih sličic. Program zato ne sme spreminjati grafične vsebine, medtem ko se zaslonska slika osvežuje. Grafična kartica mora zato obvestiti CPU, kdaj je osveževanje zaslona končano, kar pomeni, da CPU lahko začne s spreminjanjem grafične vsebine, seveda pa mora z izrisovanjem sličice zaključiti pred začetkom naslednjega osveževanja. Ta signal se imenuje *VSync* (vertical synchronization), proži pa ga grafična kartica preko zahteve za prekinitev. Slika 9.2 prikazuje časovnico sinhroniziranega kreiranja zaporednih sličic animacije.



Slika 9.2: Sinhronizacija med CPE in računalniškim zaslonom.

Program, ki prikazuje neko grafično animacijo, je tipično strukturiran na naslednji način:

1. Preberi uporabniški odziv (če je upoštevan).
2. Izriši sličico (*frame*).
3. Počakaj na dogodek VSync.
4. Ponovi za naslednjo sličico.

Naš simuliran grafični zaslon ima frekvenco osveževanja 50 Hz, kar pomeni, da naše animacije (in igre) tečejo s frekvenco 50 sličic na sekundo (angl. *frames per second*, FPS). Bodite pozorni, da svoj program poganjate ob dovolj visoki frekvenci CPE, tako da lahko posamezno sličico izriše v največ 20 milisekundah. Opazili boste, da ko je program sinhroniziran s signalom VSync, frekvenca CPE na hitrost animacije nima vpliva, v kolikor je CPE dovolj hiter, da lahko sličice producira pravočasno.

Task 9.8 Na položaju $(7, 0)$ izrišite znak, ki ima ASCII kodo 255 (krožec). Znak naj se potem vsake $0,5$ sekunde (25 sličic) spusti za eno vrstico. Ko doseže spodnjo vrstico, naj se postavi nazaj na položaj $(7, 0)$. Program naj po začetnem izrisu krožca animacijo zakasni za dve sekundi.

Kateri del vašega programa bi potencialno lahko povzročil učinek utripanja, če ne bi bil pravilno sinhroniziran s signalom VSync?

Rešitev:

```
JMP main
JMP isr

; The vertical sync signal.
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.

; 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 up the graphics screen.
    MOV A, 1              ; Set the graphics card to TEXT mode
    OUT 7                 ; through register VIDMODE.

    ; Initialize the object position.
    MOV D, 0x000E         ; Initial cell is (7, 0).
    MOV A, D              ; Activate the cell
    OUT 8                 ; through VIDADDR.
    MOVB AH, 255          ; The right shape.
    MOVB AL, 255          ; White color.
    OUT 9                 ; Print the character.

    ; Enable graphics card interrupts.
    MOV A, 4              ; Mask the graphics card interrupts.
    OUT 0                 ; Enable the graphics card interrupts.
    STI                   ; Enable interrupts globally.

    ; Wait for 2 seconds before the animation starts.
    MOV C, 100
    CALL wait_frames      ; Call wait_frames(100).

    ; Enter the animation loop.
loop:
    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 object.
    ADD D, 256        ; Move one line down.
    CMPB DH, 16       ; Is the object in the 16-th line?
    JB noreset        ; If not, do not reset its position.
    MOV D, 0x000E     ; Otherwise put it back to cell (7, 0).
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 C, 25         ; Wait for 25 frames.
    CALL wait_frames  ; Call wait_frames(25).
    JMP loop          ; Repeat for the next frame.
    HLT
```

Če sinhronizacija s signalom VSync ne bi bila izvedena pravilno, bi se zaslon lahko osvežil ravno takrat,

ko bi CPE izvajala ukaze v vrsticah 61 – 69. Slika je v tem trenutku prazna, saj je objekt že bil izbrisan s prejšnjega položaja, nov položaj pa še ni izračunan. To bi povzročilo prikaz prazne sličice in videti bi bilo, kot da je naš objekt utripnil.

Task 9.9 Izpišite niz "The Light at the End of the World", z izpisovanjem pa pričnite v celici $(16, 7)$. Besedilo naj bo rumene barve, ozadje zaslona pa naj bo modro. Besedilo je na začetku očitno izven vidnega polja. Nato premikajte prikazno okno horizontalno, za eno piko po vsakem signalu VSync. To bo povzročilo učinek gladkega drsenja besedila od desne proti levi. Ko besedilo popolnoma izgine, postavite odmik prikaznega okna nazaj na 0 ter to zanko ponavljajte v neskončnost.

Rešitev:

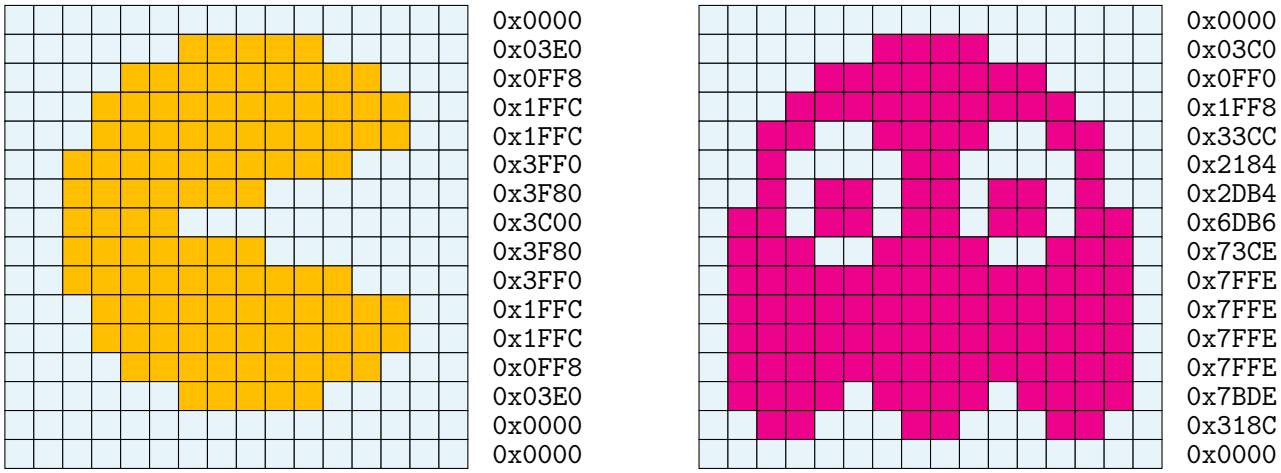
```
1  JMP main
2  JMP isr
3
4  ; The vertical sync signal.
5  vsync: DW 0
6
7  ; The ISR to serve the graphics card interrupt requests.
8  isr:
9      PUSH A          ; ISR will use register A.
10     MOV [vsync], 1  ; Set the vsync flag.
11     MOV A, 4        ; Clear the graphics card interrupt request
12     OUT 2           ; through the I/O register IRQEOI.
13     POP A           ; Restore the original value of register A.
14     IRET            ; Return from interrupt.
15
16 str: DB "The Light at the End of the World"
17      DB 0
18
19 ; Function wait_next_frame() waits for the vsync signal.
20 wait_next_frame:
21     PUSH A                   ; The function will use register A.
22 wait_next_frame_loop:
23     MOV A, [vsync]           ; Check the current vsync value.
24     CMP A, 0                 ; If still 0,
25     JE wait_next_frame_loop  ; check it again.
26     MOV [vsync], 0           ; The vsync signal received. Reset it.
27     POP A                    ; Restore the original value of register A.
28     RET
29
30 main:
31     MOV SP, 0x0FFF       ; Initialize the stack pointer.
32
33     ; Set up the graphics mode.
34     MOV A, 1             ; Set the graphics card to TEXT mode
35     OUT 7                ; through register VIDMODE.
36     MOV A, 0xA300        ; The VRAM address of the background color.
```

```
37      OUT 8                   ; Activate the background color address.
38      MOV A, 3                ; Set the background color to blue
39      OUT 9                   ; through the VIDDATA register.
40
41      ; Print out the string.
42      MOV C, str              ; Pointer to string str.
43      MOV D, 0x0720           ; Pointer to the graphics cell at (16, 7).
44  print_loop:
45      MOV A, D                ; Activate the graphics cell
46      OUT 8                   ; through the VIDADDR register.
47      MOVB AH, [C]            ; Get the current string character.
48      CMPB AH, 0              ; If zero, we are done printing.
49      JE print_break
50      MOVB AL, 252            ; Set up the yellow color.
51      OUT 9                   ; Print the character.
52      INC C                   ; Next character.
53      ADD D, 2                ; Next screen cell.
54      JMP print_loop          ; Repeat for the next character.
55  print_break:
56
57      ; Enable graphics card interrupts.
58      MOV A, 4                ; Mask the graphics card interrupts.
59      OUT 0                   ; Enable the graphics card interrupts.
60      STI                     ; Enable interrupts globally.
61
62      ; Scroll the screen.
63      MOV A, 0xA302           ; The VRAM address of the HScroll information.
64      OUT 8                   ; Activate the HScroll VRAM address.
65      MOV A, 0                ; Let the offset be initially 0.
66  offset_loop:
67      OUT 9                   ; Set the current offset.
68      INC A                   ; Prepare the new offset value.
69      CMP A, 784              ; If the offset has reached 784 pixels,
70      JB offset_continue
71      MOV A, 0                ; reset the offset back to 0.
72  offset_continue:
73      CALL wait_next_frame ; Wait for the next frame.
74      JMP offset_loop
75      HLT
```

Priprava na izpit:

a) Razširite rešitev naloge 9.8 tako, da sta sedaj prikazana dva krožca, ki se premikata v nasprotnih smereh, torej da se eden premika navzgor, drugi pa navzdol.

b) Razširite rešitev naloge 9.9 tako, da lahko uporabnik preko tipkovnice spreminja smer in hitrost drsenja besedila.



Slika 9.3: Točkovne definicije figure Pacman-a in duhcev.

9.4 Spreminjanje oblik in barv

Preden grafična kartica v besedilnem načinu izriše nek znak v neki barvi, definicijo te oblike in barve poišče v VRAM-u. Vsak znak je sestavljen iz 16×16 pik. Vsaka pika pa je lahko bodisi vidna bodisi nevidna, zato lahko v en zlog zapišemo informacijo o osmih pikah. Za definicijo oblike enega znaka tako potrebujemo 32 zlogov. V razdelku A.7.1 lahko vidimo primer takšne definicije za črko 'A'. Definicije znakov se v VRAM-u začnejo na naslovu `0x8000` (desetiško 32768) in zavzemajo $256 \cdot 32$ B $= 8192$ B $= 8$ KB. Če želimo, na primer, dostopati do definicije črke 'A', ki ima zaporedno številko 65 v tabeli ASCII, njen naslov v VRAM-u lahko izračunamo takole: $32768 + 65 \cdot 32 = 34848 =$ `0x8820`.

Definicije barv se začnejo na naslovu `0xA000` (desetiško 40960), kjer najdemo 24-bitne vnose RGB za vsakega od 256 barvnih indeksov. Za dostop do definicije barve z indeksom 252 bi njen naslov izračunali takole: $40960 + 252 \cdot 3 = 41716 =$ `0xA2F4`. Na tem naslovu bi našli 8-bitno rdečo barvno komponento (255), sledili pa bi zelena (255) in modra (0) vrednost na naslovih `0xA2F5` in `0xA2F6`. Tako je definirana rumena barva z indeksom 252 v barvni paleti 3-3-2.

Znake in barve lahko kadarkoli spremenimo, preprosto tako, da spremenimo vsebino v VRAM-u. Učinki so vidni takoj oz. ob naslednjem izrisu slike. Običajna praksa preoblikovanja znakov je takšna, da definicije v VRAM-u preprišemo z novimi vrednostmi iz RAM-a. Za primer si oglejmo dva lika iz igre Pacman, ki sta prikazani na sliki 9.3. Statično polje (angl. *static array*), ki vsebuje lik Pacman-a, bi bilo v zbirnem jeziku definirano kot:

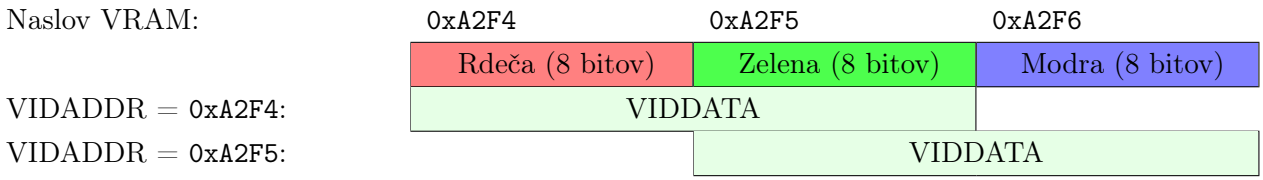
```
DB "\x00\x00\x03\xE0\x0F\xF8\x1F\xFC\x1F\xFC\x3F\xF0\x3F\x80\x3C\x00
    \x3F\x80\x3F\xF0\x1F\xFC\x1F\xFC\x0F\xF8\x03\xE0\x00\x00\x00\x00"
```

lik duhca pa kot:

```
DB "\x00\x00\x03\xC0\x0F\xF0\x1F\xF8\x33\xCC\x21\x84\x2D\xB4\x6D\xB6
    \x73\xCE\x7F\xFE\x7F\xFE\x7F\xFE\x7F\xFE\x7B\xDE\x31\x8C\x00\x00"
```

Celotno barvno paleto lahko spremenimo na podoben način, pri spreminjanju ene same barve pa moramo biti pazljivi. Ponovno si poglejmo, kako dostopamo do rumene barve z indeksom 252. Najprej v register

VIDADDR zapišemo naslov `0xA2F4` in tako dostopamo do zgornjega 16-bitnega dela 24-bitne definicije barve. To pomeni, da hkrati dostopamo do rdeče in zelene komponente. Da bi dosegli modro komponento, nastavimo register VIDADDR na naslov `0xA2F4`, s tem pa dobimo dostop do zelene in modre komponente.



Task 9.10 Na grafični zaslon izpišite niz "ABBA", nato pa spremenite obliko črke 'A' v obliko duhca s slike 9.3.

Rešitev:

```
1  JMP main
2
3  str:
4      DB "ABBA\x00"
5
6  tile_ghost:
7      DB "\x00\x00\x03\xC0\x0F\xF0\x1F\xF8\x33\xCC\x21\x84\x2D\xB4\x6D\xB6"
8      DB "\x73\xCE\x7F\xFE\x7F\xFE\x7F\xFE\x7F\xFE\x7B\xDE\x31\x8C\x00\x00"
9
10 main:
11     ; Initialize the graphics mode.
12     MOV A, 1            ; Enable the text mode
13     OUT 7               ; through I/O register VIDMODE.
14
15     ; Print the string.
16     MOV C, str          ; Pointer to string.
17     MOV D, 0            ; Pointer to text cell.
18 str_loop:
19     MOV A, D            ; Activate the current text cell
20     OUT 8               ; through I/O register VIDADDR.
21     MOVB AH, [C]        ; Get the current character.
22     CMPB AH, 0          ; If at the end of the string,
23     JE str_break        ; break the loop.
24     MOVB AL, 227        ; Set the magenta color.
25     OUT 9               ; Print the character.
26     INC C               ; Next character.
27     ADD D, 2            ; Next text cell.
28     JMP str_loop
29 str_break:
30
31     ; Redefine letter 'A'.
32     MOV C, tile_ghost ; Pointer to ghost definition data.
```

```
    MOV D, 0x8820      ; Pointer to VRAM address for character 'A'.
    MOV B, 16          ; The size of definition data in words (16-bit).
tile_set_loop:
    CMP B, 0           ; If all data copied,
    JE tile_set_break  ; break the loop.
    MOV A, D           ; Activate the VRAM address
    OUT 8              ; through I/O register VIDADDR.
    MOV A, [C]         ; Get the current word (16-bit)
    OUT 9              ; copy it to VRAM.
    DEC B              ; Decrease word counter.
    ADD C, 2           ; Next word in RAM.
    ADD D, 2           ; Next VRAM address.
    JMP tile_set_loop
tile_set_break:
    HLT
```

Task 9.11 Na naključna mesta vidnega zaslona izpišite 100 črk 'A' v rdeči barvi (indeks 224). Nato spremenite definicijo barvnih komponent te barve v R = 255, G = 0, B = 255. Črke naj bi se tako avtomatično prebarvale v vijolično barvo.

Rešitev:

Eden od izzivov te naloge je generirati naključen naslov VRAM, ki naslavlja vidno celico na zaslonu. Najprej generiramo naključno 16-bitno število, tako da preberemo vrednost V/I registra RNDGEN. To nato pretvorimo v ustrezen naslov, tako da ga maskiramo z masko:

	indeks vrstice								zlog stolpca								
A:	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	= 0x0F1E
				0 – 15								0 – 30 (soda števila)					

Maska izniči zgornje štiri bite indeksa vrstice in ga tako omeji na maksimalno število 15. Ker vsaka celica zavzema dva zloga, je veljaven zlog stolpca sodo število med 0 in 30. Z izničejem zgornjih treh bitov bi maska zlog stolpca omejila na maksimalno število 31. Ker pa izniči tudi zadnji bit, so možna le soda števila na tem intervalu.

```
JMP main

main:
    ; Initialize the graphics mode.
    MOV A, 1           ; Enable the text mode
    OUT 7              ; through I/O register VIDMODE.

    ; Print 100 randomly placed characters 'A'.
    MOV C, 100         ; Count the printed characters.
char_loop:
    CMP C, 0           ; Have we printed all 100 characters?
    JE char_break      ; If yes, break the loop.
```

```
    IN 10              ; Get a random VRAM address.
    AND A, 0x0F1E      ; Mask out a valid 16 x 16 text cell address.
    OUT 8              ; Activate the chosen text cell.
    MOV A, 0x41E0      ; Character 'A' in red color.
    OUT 9              ; Print the character.
    DEC C              ; Count the character.
    JMP char_loop
char_break:

    ; Change the definition of the red color.
    MOV A, 0xA2A0      ; VRAM address of color with index 224.
    OUT 8              ; Access the red and the green component.
    MOV A, 0xFF00      ; Set red = 255, green = 0.
    OUT 9              ; Update the red and the green component.
    MOV A, 0xA2A1      ; The lower part of the color with index 224.
    OUT 8              ; Access the green and the blue component.
    MOV A, 0x00FF      ; Set green = 0, blue = 255.
    OUT 9              ; Update the green and the blue component.
    HLT
```

Priprava na izpit:

a) Na naključna mesta postavite Pacman-a in štiri duhce. Duhci so: Blinky (RGB: 255, 0, 0), Pinky (RGB: 255, 184, 255), Inky (RGB: 0, 255, 255) in Clyde (RGB: 255, 184, 82).

b) Izrišite dva duhca, enega v vijolični, drugega pa v rumeni barvi. Rumeni duhec naj počasi zbledi in izgine. Animacijo izginjanja sinhronizirajte s signalom VSync.

9.5 Objekti (sprite)

Glavna pomanjkljivost besedilnega načina je omejitev položaja grafičnih objektov na celice. Če želimo na ta način animirati nek objekt, je njegovo gibanje diskretno, saj se lahko premakne le iz ene celice v drugo. To pomanjkljivost lahko ublažimo z dodatno funkcionalnostjo besedilnega načina, ki je znana kot *sprite* grafika. *Sprite* je neodvisen grafični objekt (črka oz. znak), ki ga lahko postavimo na poljubno točko (pixel) na zaslonu. Takšen objekt je popolnoma neodvisen od postavitev celic in njihove vsebine, prav tako pa tudi od premikanja vidnega dela zaslona (scrolling). Objekti *sprite* so vedno prikazani v ospredju, celice z besedilom pa v ozadju. V računalniških igrah so objekti *sprite* običajno prosto gibajoči se objekti (bitja), okolica pa je izrisana v celicah.

Različni sistemi podpirajo različno število in velikosti sprite-ov. Naša grafična kartica dopušča do osem sprite-ov v velikosti 16×16 pik. V našem sistemu je sprite torej zgolj še en ASCII znak, iz istega nabora ASCII oblik in barv, iz katerega je tudi vsebina celic. Oblike sprite-ov lahko zato poljubno spreminjamo skupaj z vsemi ostalimi znaki. Informacije o sprite-ih se začnejo na VRAM naslovu `0xA306`, vsak sprite pa zavzame 4 zloge (koda ASCII, indeks barve, položaj x, položaj y). Vse štiri vrednosti se lahko gibljejo med 0 in 255. Na začetku je ASCII koda pri vseh osmih sprite-ih nastavljena na 0, kar pomeni, da je sprite izključen.

Task 9.12 Opazujmo, kako objekti sprite delujejo.

1. Naj bo sprite 1 rdeča črka 'A'. Postavite ga na položaj (3px, 3px).
2. Prepričajmo se, da je položaj sprite-a res neodvisen od celic pod njim. V celico (0, 0) izpišite belo črko 'A'. Opazili boste, da se obe črki prekrivata le delno, kar pomeni, da se sprite nahaja med dvema celicama.
3. Prepričajmo se, da je položaj sprite-a res neodvisen od premikanja vidnega dela zaslona. Premaknite vidno okno za osem pik v desno. Opazili boste, da se beli 'A' (celica) premakne levo, rdeči 'A' (sprite) pa ostane na mestu (3px, 3px).

Rešitev:

```
; Enable graphics card.
MOV A, 1       ; Set the text mode.
OUT 7          ; VIDMODE = 1

; Place a sprite 'A' to screen position (3px, 3px).
MOV A, 0xA306 ; VRAM address of sprite 1.
OUT 8          ; Activate the address.
MOV A, 0x41E0 ; Let sprite 1 be a red letter 'A'.
OUT 9          ; Set the sprite shape/color.
MOV A, 0xA308 ; VRAM address of sprite 1 screen position.
OUT 8          ; Activate the address.
MOV A, 0x0303 ; Let sprite 1 be at (3px, 3px).
OUT 9          ; Set the sprite position.

; Draw a white letter 'A' to text cell (0, 0).
MOV A, 0       ; VRAM address of text cell (0, 0).
OUT 8          ; Activate the cell.
MOV A, 0x41FF ; White letter 'A'.
OUT 9          ; Place the letter to text cell (0, 0).

; Scroll the screen horizontally by 8 pixels.
MOV A, 0xA302 ; VRAM address of horizontal screen offset.
OUT 8          ; Activate the address.
MOV A, 8       ; Move by 8 pixels.
OUT 9          ; Set the new offset.
HLT
```

Task 9.13 Ponovno si oglejte rešitev naloge 9.8. Spomnite se, da smo pri tej nalogi animirali padanje objekta, celico po celico. Spremenite ta program tako, da bo objekt padal zvezno, po eno piko vsako sličico (frame). Dodajte še nekaj besedila, tako da bo objekt padel skozenj.

Rešitev:

```
JMP main
JMP isr
```

```
3
4  ; The vertical sync signal.
5  vsync: DW 0
6
7  ; The ISR to serve the graphics card interrupt requests.
8  isr:
9      PUSH A                  ; ISR will use register A.
10     MOV [vsync], 1          ; Set the vsync flag.
11     MOV A, 4                ; Clear the graphics card interrupt request
12     OUT 2                   ; through the I/O register IRQEOI.
13     POP A                   ; Restore the original value of register A.
14     IRET                    ; Return from interrupt.
15
16 ; Function wait_frames(count) waits for count frames.
17 ; Parameter count is given through register C.
18 wait_frames:
19 wait_frames_loop:
20     MOV A, [vsync]          ; Check the current vsync value.
21     CMP A, 0                ; If still 0,
22     JE wait_frames_loop     ; check it again.
23     MOV [vsync], 0          ; The vsync signal received. Reset it.
24     DEC C                   ; Count the received frame.
25     CMP C, 0                ; When enough frames passed,
26     JE wait_frames_return   ; return from the function.
27     JMP wait_frames_loop    ; Otherwise, wait another frame.
28 wait_frames_return:
29     RET
30
31 ; Some text to appear behind the falling object.
32 str: DB "Systems 1"
33      DB 0
34
35 main:
36     MOV SP, 0x0FFF          ; Initialize the stack pointer.
37
38     ; Set up the graphics screen.
39     MOV A, 1                ; Set the graphics card to TEXT mode
40     OUT 7                   ; through register VIDMODE.
41
42     ; Print the string.
43     MOV C, str              ; Pointer to string.
44     MOV D, 0x0808           ; Pointer to text cell.
45 str_loop:
46     MOV A, D                ; Activate the current text cell
47     OUT 8                   ; through I/O register VIDADDR.
48     MOVB AH, [C]            ; Get the current character.
49     CMPB AH, 0              ; If at the end of the string,
50     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.

    ; 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.

    ; Enable graphics card interrupts.
    MOV A, 4              ; Mask the graphics card interrupts.
    OUT 0                 ; Enable the graphics card interrupts.
    STI                   ; Enable interrupts globally.

    ; 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.
    MOV C, 1              ; Wait the next frame.
    CALL wait_frames      ; Call wait_frames(1).
    JMP loop              ; Repeat for the next frame.
    HLT
```

Priprava na izpit:

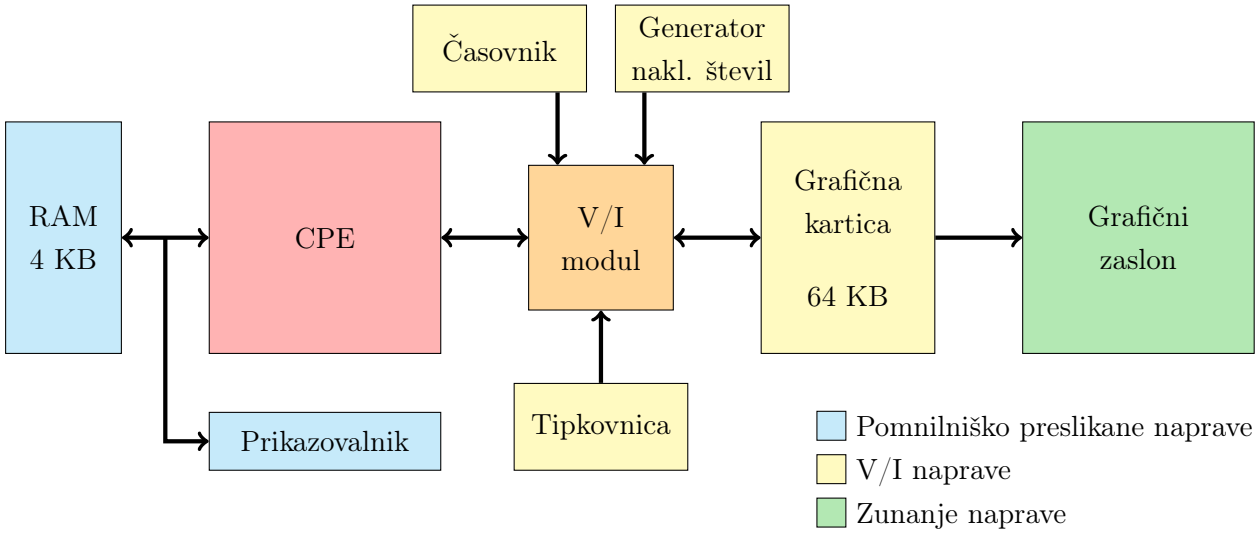
a) Rešitvi naloge 9.13 dodajte še drugi sprite, ki se premika navzgor z nekoliko drugačno hitrostjo od prvega.

b) Spremenite obliko objekta pri nalogi 9.13 v obliko duhca s slike 9.3.

c) Pri nalogi 9.13 animirajte tudi besedilo, tako da vidno okno premikate po eno piko na vsako sličico (frame).

A Arhitektura simuliranega sistema

A.1 Arhitektura sistema



A.2 Registri CPE

16-bitni registri:

Register	Opis	Indeks
A	Splošno-namenski register	0
B	Splošno-namenski register	1
C	Splošno-namenski register	2
D	Splošno-namenski register	3
SP	Kazalec na sklad	4
IP	Kazalec na ukaz	5
SR	Statusni register	6

8-bitni registri:

Register	Opis	Indeks
AH	Višji del registra A	7
AL	Nižji del registra A	8
BH	Višji del registra B	9
BL	Nižji del registra B	10
CH	Višji del registra C	11
CL	Nižji del registra C	12
DH	Višji del registra D	13
DL	Nižji del registra D	14

A.3 Načini naslavljanja

Način naslavljanja	Okrajšava	Zgled
Takojšnje (immediate)	IMD	`ADD A, 100`
Registrsko (register)	REG	`ADD A, B`
Neposredno (direct)	DIR	`ADD A, [100]`
Posredno (indirect)	IND	`ADD A, [B]`

A.4 Nabor ukazov

Format ukaza CPE:

operacijska koda (opcode)	operand 1 (neobvezno)	operand 2 (neobvezno)

Nabor ukazov CPE:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	HLT	MOV REG,REG	MOV REG,IND	MOV REG,DIR	MOV IND,REG	MOV DIR,REG	MOV REG,IMD	MOV IND,IMD	MOV DIR,IMD	MOVB REG,REG	MOVB REG,IND	MOVB REG,DIR	MOVB IND,REG	MOVB DIR,REG	MOVB REG,IMD	MOVB IND,IMD
1	MOVB DIR,IMD	ADD REG,REG	ADD REG,IND	ADD REG,DIR	ADD REG,IMD	ADDB REG,REG	ADDB REG,IND	ADDB REG,DIR	ADDB REG,IMD	SUB REG,REG	SUB REG,IND	SUB REG,DIR	SUB REG,IMD	SUBB REG,REG	SUBB REG,IND	SUBB REG,DIR
2	SUBB REG,IMD	INC REG	INCB REG	DEC REG	DECB REG	CMP REG,REG	CMP REG,IND	CMP REG,DIR	CMP REG,IMD	CMPB REG,REG	CMPB REG,IND	CMPB REG,DIR	CMPB REG,IMD	JMP IND	JMP DIR	JC IND
3	JC DIR	JNC IND	JNC DIR	JZ IND	JZ DIR	JNZ IND	JNZ DIR	JA IND	JA DIR	JNA IND	JNA DIR	PUSH REG	PUSH IMD			PUSHB REG
4	PUSHB IMD			POP REG	POPB REG	CALL IND	CALL DIR	RET	MUL REG	MUL IND	MUL DIR	MUL IMD	MULB REG	MULB IND	MULB DIR	MULB IMD
5	DIV REG	DIV IND	DIV DIR	DIV IMD	DIVB REG	DIVB IND	DIVB DIR	DIVB IMD	AND REG,REG	AND REG,IND	AND REG,DIR	AND REG,IMD	ANDB REG,REG	ANDB REG,IND	ANDB REG,DIR	ANDB REG,IMD
6	OR REG,REG	OR REG,IND	OR REG,DIR	OR REG,IMD	ORB REG,REG	ORB REG,IND	ORB REG,DIR	ORB REG,IMD	XOR REG,REG	XOR REG,IND	XOR REG,DIR	XOR REG,IMD	XORB REG,REG	XORB REG,IND	XORB REG,DIR	XORB REG,IMD
7	NOT REG	NOTB REG	SHL REG,REG	SHL REG,IND	SHL REG,DIR	SHL REG,IMD	SHLB REG,REG	SHLB REG,IND	SHLB REG,DIR	SHLB REG,IMD	SHR REG,REG	SHR REG,IND	SHR REG,DIR	SHR REG,IMD	SHRB REG,REG	SHRB REG,IND
8	SHRB REG,DIR	SHRB REG,IMD	CLI	STI	IRET			IN REG	IN IND	IN DIR	IN IMD	OUT REG	OUT IND	OUT DIR	OUT IM	

A.5 Pomnilniški prostor

Razpon naslovov	Komponenta
`0x0000` – `0x0FFF`	RAM (4 KB)
`0x1000` – `0x101F`	Prikazovalnik (2 vrstici × 16 znakov ASCII)

A.6 Vhodno / izhodni modul

I/O Registri:

Register	Opis	Indeks
IRQMASK	Omogoči/onemogoči specifične zahteve za prekinitev	0
IRQSTATUS	Trenutno zahtevane prekinitve	1
IRQEIO	Počisti specifično zahtevo za prekinitev	2
TMRPRELOAD	Začetna vrednost časovnika	3
TMRCOUNTER	Trenutna vrednost časovnika	4
KBDSTATUS	Status tipkovnice (zaznan je pritisk tipke)	5
KBDDATA	Podatek, ki je bil prejet od tipkovnice	6
VIDMODE	Način grafične kartice	7
VIDADDR	Naslov v grafičnem pomnilniku VRAM	8
VIDDATA	Podatek na naslovu VIDADDR	9
RNDGEN	Naključno generirano število	10

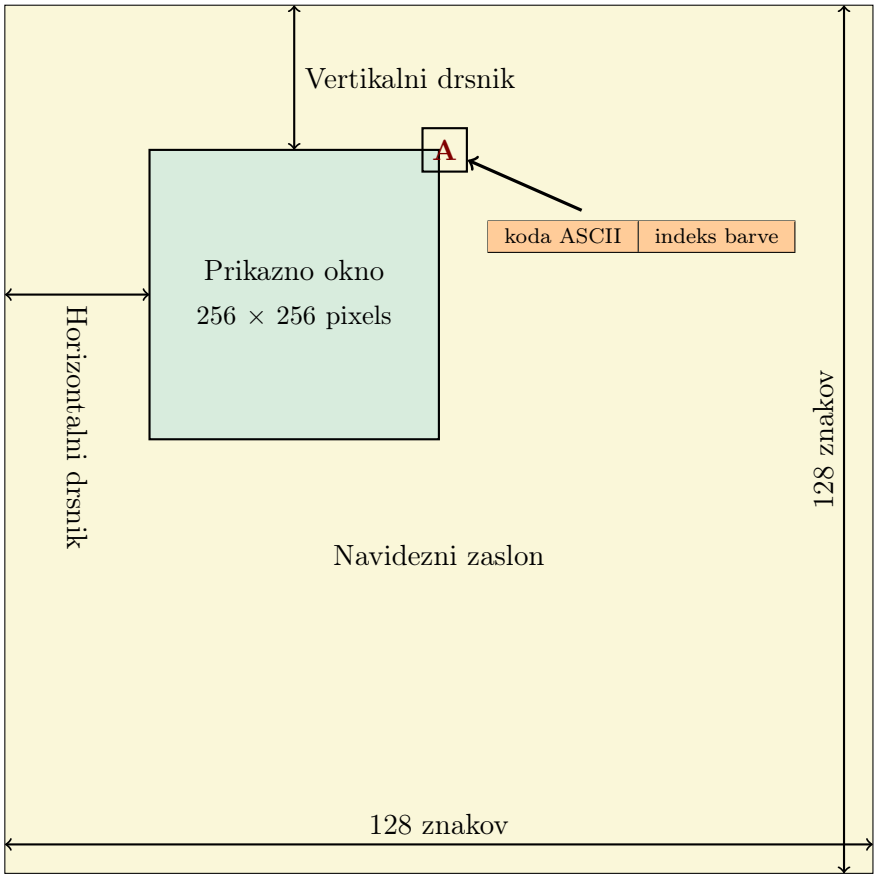
A.7 Grafična kartica

VIDMODE	Grafični način	Opis
0	DISABLED	Grafična kartica je onemogočena (privzeto).
1	TEXT	Prikazovanje besedila in novo-definiranih znakov.
2	BITMAP	Prikazovanje individualnih pik.
3	CLEAR	Pobriše zaslon. Deluje v obeh grafičnih načinih.
4	RESET	Ponastavi in onemogoči grafično kartico.

A.7.1 Besedilni način (VIDMODE = 1)

Značilnosti besedilnega grafičnega načina:

- Velikost vsakega znaka je 16 × 16 pik.
- Vsak prikazan znak ima dve lastnosti: koda ASCII (8-bitov) in indeks barve (8-bitov).
- Privzete definicije oblik vseh 256 znakov se nahajajo na VRAM naslovih `0x8000` – `0x9FFF` in jih je možno prepisati. Privzete vrednosti so ponastavljene ob ponastavitvi grafične kartice.
- Standardna paleta RRRGGGBB se nahaja na VRAM naslovih `0xA000` – `0xA2FF` (3 zlogi/barvo) in jo je možno prepisati. Privzete vrednosti so ponastavljene ob ponastavitvi grafične kartice.
- Velikost navideznega zaslona je 128 × 128 znakov, toda vidno je le prikazno okno velikosti 256 × 256 pik. Prikazno okno lahko premikamo in tako dosežemo učinek gladkega drsenja.
- Osem znakov lahko postavimo na poljubno lokacijo (256 × 256) prikaznega okna, neodvisno od vsebine virtualnega zaslona ter lokacije prikaznega okna (t. i. *sprite* grafika).



Uporaba pomnilnika VRAM v besedilnem načinu:

Lokacija v VRAM	Velikost	Uporaba
`0x0000` – `0x7FFF`	32 KB	Področje navideznega zaslona (128 × 128 znakov, 2 zloga/znak).
`0x8000` – `0x9FFF`	8 KB	Definicje oblik znakov (256 znakov, 32 zlogov/znak).
`0xA000` – `0xA2FF`	0.75 KB	Barvna paleta (256 barv, 3 zlogi/barvo).
`0xA300` – `0xA301`	2 B	Barva ozadja (indeks barve 0 – 255, naslov `0xA300` ni uporabljen).
`0xA302` – `0xA303`	2 B	Horizontalni drsnik (odmik prikaznega okna v smeri x v pikah).
`0xA304` – `0xA305`	2 B	Vertikalni drsnik (odmik prikaznega okna v smeri y v pikah).
`0xA306` – `0xA309`	4 B	Sprite 1 (znak, barva, x, y).
`0xA30A` – `0xA30D`	4 B	Sprite 2 (znak, barva, x, y).
`0xA30E` – `0xA311`	4 B	Sprite 3 (znak, barva, x, y).
`0xA312` – `0xA315`	4 B	Sprite 4 (znak, barva, x, y).
`0xA316` – `0xA319`	4 B	Sprite 5 (znak, barva, x, y).
`0xA31A` – `0xA31D`	4 B	Sprite 6 (znak, barva, x, y).
`0xA31E` – `0xA321`	4 B	Sprite 7 (znak, barva, x, y).
`0xA322` – `0xA325`	4 B	Sprite 8 (znak, barva, x, y).
`0xA326` – `0xFFFF`		Prost pomnilnik (23754 zlogov).

Podatki o obliki znaka (32 zlogov):

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0x1FF8
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0x3FFC
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0x3C3C
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0X3C3C
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0x3C3C
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0x3FFC
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0x3FFC
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0x3C3C
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0x3C3C
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0x3C3C
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0x3C3C
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0x3C3C
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000

Področje navideznega zaslona:

znak 0	ASCII koda	indeks barve	32 KB
znak 1	ASCII koda	indeks barve	
⋮	⋮		
znak n	ASCII koda	indeks barve	

Definicija nabora znakov:

ASCII koda 0	podatki oblike 0	8 KB
ASCII koda 1	podatki oblike 1	
⋮	⋮	
ASCII koda 255	podatki oblike 255	

Definicija barvne palete:

barva 0	red (8-bit)	green (8-bit)	blue (8-bit)	765 bytes
barva 1	red (8-bit)	green (8-bit)	blue (8-bit)	
⋮	⋮			
barva 255	red (8-bit)	green (8-bit)	blue (8-bit)	

Podatki o prostih oblikah (*sprite*) :

sprite 1	ASCII koda	barva	x	y	32 bytes
sprite 2	ASCII koda	barva	x	y	
⋮	⋮				
sprite 8	ASCII koda	barva	x	y	

A.7.2 Točkovni način (VIDMODE = 2)

Vseh 64 KB grafičnega pomnilnika VRAM je uporabljenega za prikaz rastrske slike (256 × 256 pik). Uporabljena je standardna 8-bitna barvna paleta RRRGGGBB, ki pa je ni mogoče spremeniti.