ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1992

Seria: INFORMATYKA z. 20 Nr kol. 1190

Jarosław SKOLIMOWSKI

REALIZACJA OPERACJI GRAFICZNYCH NISKIEGO POZIOMU DLA KARTY EGA

Streszczenie. W pracy przedstawiono charakterystyczne cechy architektury karty sterownika wizji EGA oraz przykłady elementarnych procedur graficznych w języku C dla trybów 16-kolorowych wysokiej rozdzielczości, odwołujących się bezpośrednio do pamięci obrazu i rejestrów wewnętrznych tej karty.

LOW LEVEL GRAPHICS ROUTINES FOR EGA CARD

Summary. A characteristic features of EGA card architecture and implementation of register-level graphics routines in "C" language are presented.

LES PLUS BAS NIVEAU PROCEDURES GRAPHIQUES POUR UNE CARTE EGA

Resumé. L'article present les divers procedures graphiques pour une carte EGA. Les procedures implementées en 'C' et crées au plus bas niveau (niveau registre) de controleur video.

1. Wprowadzenie

Każda złożona procedura graficzna, generująca obraz na monitorze lub innym wyjściowym urządzeniu graficznym komputera, sprowadza się do cyklicznego, określonego konkretnym algorytmem, wykonywania operacji podstawowych, komunikujących się bezpośrednio ze sprzętem. Na efektywność działania programu graficznego wpływ ma za-

równo dobór algorytmu generacji tych wywołań, jak i efektywność wykonania pojedynczych operacji elementarnych.

Dla standardowych sterowników wizji komputerów IBM PC dostępne są systemowe funkcje graficzne (przerwanie nr 10h BIOS-a) oraz biblioteki procedur graficznych popularnych kompilatorów języków programowania (TURBO-C, TURBO-PASCAL itd.). Procedury BIOS-a są nieliczne i stosunkowo wolne. Procedury standardowe języków programowania z reguły omijają BIOS, dzięki czemu są szybsze, nie wykorzystują jednak wszystkich możliwości sprzętu. W przypadku tworzenia aplikacji graficznej, dla której czas jest czynnikiem krytycznym, konieczne jest samodzielne programowanie na poziomie komórek pamięci VRAM i rejestrów wewnętrznych sterownika wizji. W pracy przedstawiono krótki opis architektury karty EGA oraz kilka przykładów wykorzystania jej charakterystycznych cech do przyspieszenia operacji graficznych.

2. Architektura karty EGA

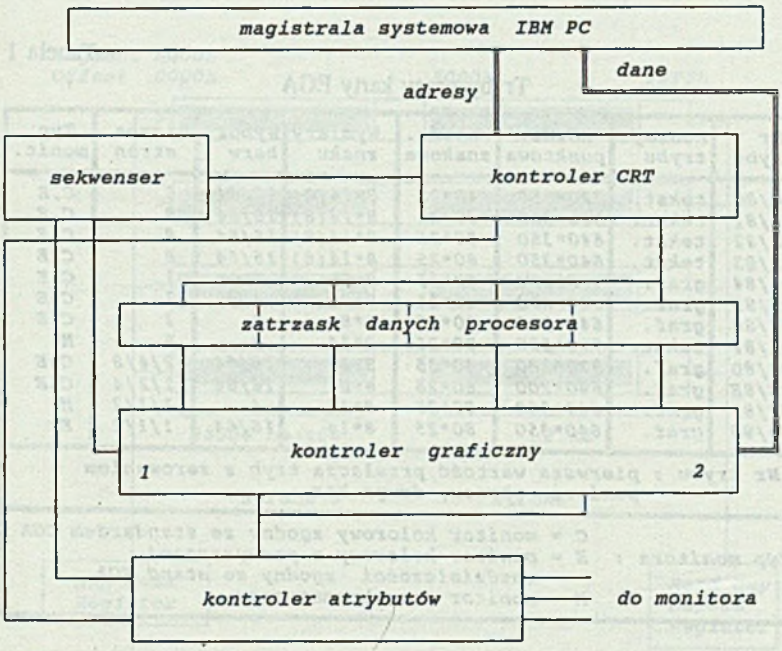
Karta EGA (Enhanced Graphics Adapter) pełni funkcję sterownika monitora ekranowego w mikrokomputerach firmy IBM. Współpracuje z ekranem w trybach tekstowych i graficznych, zapewniając emulację starszych rozwiązań firmy: kart CGA i MDA.

Oryginalna karta EGA umożliwia pracę z rozdzielczością 640 x 350 punktów. Liczba kolorów zależna jest od wielkości zamontowanej na płytce pamięci obrazu. Pełne wykorzystanie dostępnej palety zapewnia pamięć 128 kB, dla której można uzyskać na ekranie 16 barw wybranych dowolnie spośród 64. Minimalna wersja karty EGA zawiera tylko 64 kB RAM, co pozwala wykorzystać 4 kolory w trybie najwyższej rozdzielczości. Maksymalna rozbudowa pamięci obrazu do 256 kB, obok rozszerzenia palety barw dołącza drugi "bank" na alternatywny obraz, co stwarza możliwość płynnego przesuwania zawartości ekranu i szybką zmianę obrazu.

Karta sterownika składa się pod względem funkcjonalnym z czterech głównych układów (kontrolerów), które w wersji oryginalnej stanowią pięć układów VLSI. Są to:

- kontroler CRT – jeden układ VLSI,
- sekwenser – jeden układ VLSI,
- kontroler graficzny – dwa układy VLSI,
- kontroler atrybutów – jeden układ VLSI.

Uproszczony schemat blokowy ich połączeń przedstawiono na rysunku 1. Układy kontrolerów zawierają w sumie około siedemdziesięciu programowalnych rejestrów dostępnych dla użytkownika. Karta posiada własne podstawowe oprogramowanie wejścia/wyjścia (tzw. BIOS) zapisane w pamięci ROM. Oprogramowanie to jest instalowane jako nakładka na BIOS główny komputera poprzez modyfikację zawartości wektora przerwań w czasie zimnego startu. Procedury obsługi ekranu zawarte w BIOS karty dostępne są z poziomu komputera przez przerwanie programowe nr 10h (zapis 10h oznacza liczbę szesnastkową).



Rys. 1. Schemat blokowy karty EGA
Fig. 1. Main logic components of EGA card

2.1. Organizacja pamięci obrazu

Sposób odwzorowania ekranu monitora w pamięci obrazu zależy od aktualnie wybranego trybu pracy sterownika wizji oraz od rozmiaru fizycznie przyłączonej pamięci. Standardowe tryby pracy zestawiono w **tabeli 1.**

Wszystkie tryby, zarówno alfanumeryczne, jak i graficzne, dostępne są w dwóch wersjach opatrzonych różnymi numerami. Wersja z numerem niższym wymazuje zawartość VRAM w trakcie przełączenia, wersja z numerem wyższym pozostawia zawartość VRAM bez zmian.

Tryby 00-07 emulują działanie analogicznych trybów alfanumerycznych i graficznych kart CGA i MDA. Jedyną różnicą z punktu widzenia programowania operacji we/wy są inne adresy portów kontrolera CRT.

Tryby 0Dh-10h są nowymi trybami graficznymi. Pełne możliwości karty, tzn. rozdzielczość 640x350 i wybór 16 kolorów z palety 64 uzyskuje się w trybie 10h.

Zamieszczone w dalszej części artykułu opisy sprzętu i programowania rejestrów wewnętrznych sterownika EGA zostaną zawężone do tego przypadku.

Tabela 1

Tryby pracy karty EGA

Nr trybu	Rodzaj trybu	Rozdz. punktowa	Rozdz. znakowa	Wymiary znaku	Wybór barw	Liczba stron	Typ monit.
00/80	tekst.	320*350	40*25	8*14(8)	16/64	8	C,E
01/81	tekst.	320*350	40*25	8*14(8)	16/64	8	C,E
02/82	tekst.	640*350	80*25	8*14(8)	16/64	8	C,E
03/83	tekst	640*350	80*25	8*14(8)	16/64	8	C,E
04/84	graf.	320*200	40*25	8*8	4	1	C,E
05/85	graf.	320*200	40*25	8*8	4	1	C,E
06/86	graf.	640*200	80*25	8*8	2	1	C,E
07/87	tekst.	720*350	80*25	9*14	4	8	M
0D/8D	graf.	320*200	40*25	8*8	16/64	2/4/8	C,E
0E/8E	graf.	640*200	80*25	8*8	16/64	1/2/4	C,E
0F/8F	graf.	640*350	80*25	8*14	4	1/1/2	M
10/90	graf.	640*350	80*25	8*14	16/64	1/1/2	E

Nr trybu : pierwsza wartość przełącza tryb z zerowaniem pamięci obrazu, druga bez

Typ monitora : C = monitor kolorowy zgodny ze standardem CGA
E = monitor kolorowy o podwyższonej rozdzielczości zgodny ze stand EGA
M = monitor monochromatyczny

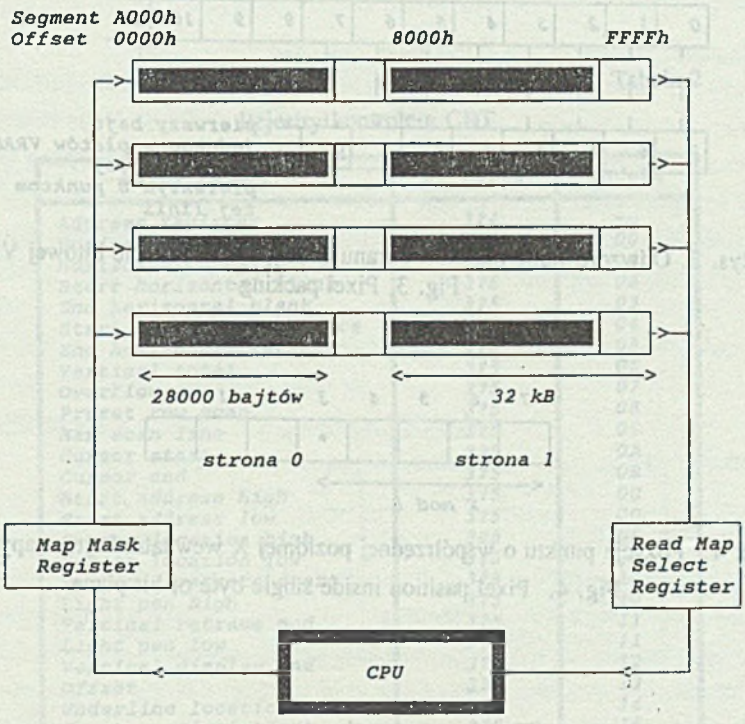
W trybie 10h pamięć obrazu jest podzielona na cztery równe bloki, zwane "płatami" (ang. planes), będące czterema mapami bitowymi ekranu, podłączonymi w przestrzeni adresowej komputera równolegle od adresu A000:0000h, jak na rysunku 2.

Każdemu punktowi ekranu odpowiada 1 bit w każdym z płatów (map bitowych) VRAM (rys.3), co daje razem 4-bitowy numer koloru. Poszczególne punkty ekranu są odwzorowywane w kolejności od lewej do prawej strony i z góry w dół bajtami o wzrastających liniowo adresach, bez przeplotu.

Jeśli **X,Y** – współrzędne pixela mierzone względem punktu (0,0) w lewym górnym rogu ekranu, to **offset = 80*Y + X** określa adres bajtu w każdej z 4 map bitowych, zawierającego bit informacji określający kolor punktu **(X,Y)**.

Natomiast **X mod 8** stanowi przesunięcie bitu informacji w prawo względem najstarszego bitu wewnątrz bajtu **(rys.4)**.

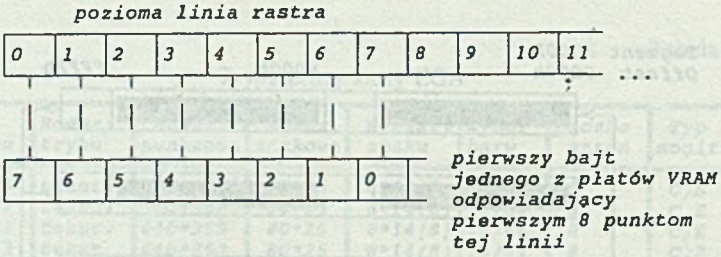
Rozmiar pojedynczego płatu stanowi zarazem wielkość przestrzeni adresowej procesora głównego zajmowanej przez kartę EGA. Dla wersji karty z 256 kB, obszar ten zawiera się od adresu A000:0000h do A000:FFFFh (2 ekrany, czyli 4 płaty po 64 kB), dla wersji 128 kB, od A000:0000h do A000:7FFFh (1 ekran 4 płaty po 32 kB), dla wersji z 64 kB, z wyjątkiem trybów 10h i 0Fh , od A000:0000 do A000:3FFFh (1 ekran 4 płaty po 16 kB).



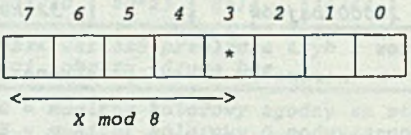
Rys. 2. Struktura pamięci obrazu
Fig. 2. EGA Video-RAM structure

W przypadku karty EGA z 64 kB VRAM w trybach 10h i 0Fh, pamięć obrazu jest podzielona nie na 4, lecz na 2 płaty po 32 kB, z których każdy podłączony jest od adresu A000:0000h do A000:7FFFh.

Dostęp do poszczególnych płatów pamięci dla operacji zapisu i odczytu ze strony procesora jest przełączany poprzez specjalne rejestry sekwensera i kontrolera graficznego. Jednorazowo możliwy jest odczyt komórki pamięci jednego z płatów, natomiast zapis równocześnie do wszystkich lub wybranych płatów. Konstrukcja sterownika zapewnia także równoczesny dostęp do pamięci obrazu ze strony procesora, jak i układów wyświetlania obrazu, co umożliwia wykonywanie operacji zapisu w dowolnym momencie bez efektu "śnieżenia" na ekranie.



Rys. 3. Odwzorowanie punktów ekranu w pojedynczej mapie bitowej VRAM
Fig. 3. Pixel packing



Rys. 4. Pozycja punktu o współrzędnej poziomej X wewnątrz bajtu mapy bitowej
Fig. 4. Pixel pasition inside single byte of bit-plane

2.2. Rejestry wewnętrzne karty EGA

Każdy z głównych podzespołów (kontrolerów) sterownika wizji zajmuje dwa adresy (porty) w przestrzeni wejścia/wyjścia procesora głównego. Każda para takich portów stanowi rejestr adresowy (indeksowy) oraz rejestr danych, służące do komunikacji z rejestrami wewnętrznymi odpowiedniego kontrolera. Zapis/odczyt rejestru wewnętrznego kontrolera odbywa się poprzez zadanie jego numeru do rejestru indeksowego i zapis /odczyt zawartości rejestru danych tego kontrolera.

2.2.1. Kontroler CRT

Steruje wejściem/wyjściem z monitora, wytwarza sygnały synchronizacji i odchylania, realizuje wyświetlanie kursora, a także zajmuje się odświeżaniem dynamicznej pamięci obrazu. Przegląd rejestrów kontrolera przedstawiono w tabeli 2. Z punktu widzenia programowania grafiki rejestry te nie są szczególnie interesujące. Wygodniejsze jest korzystanie w tym przypadku z funkcji oprogramowania BIOS. Dla niektórych zastosowań mogą być przydatne rejestry o indeksach 0Ch i 0Dh określające punkt początkowy (2-bajtowy

offset względem segmentu A000 h) w pamięci obrazu, od którego zaczyna się wyświetlanie.

Tabela 2

Rejestry kontrolera CRT

Nazwa rejestru	Nr portu	Indeks
Address register	3?4	--
Horizontal total	3?5	00
Horizontal display end	3?5	01
Start horizontal blank	3?5	02
End horizontal blank	3?5	03
Start horizontal retrace	3?5	04
End horizontal retrace	3?5	05
Vertical total	3?5	06
Overflow	3?5	07
Preset row scan	3?5	08
Max scan line	3?5	09
Cursor start	3?5	0A
Cursor end	3?5	0B
Start address high	3?5	0C
Start address low	3?5	0D
Cursor location high	3?5	0E
Cursor location low	3?5	0F
Vertical retrace start	3?5	10
Light pen high	3?5	10
Vertical retrace end	3?5	11
Light pen low	3?5	11
Vertical display end	3?5	12
Offset	3?5	13
Underline location	3?5	14
Start vertical blank	3?5	15
End vertical blank	3?5	16
Mode control	3?5	17
Line compare	3?5	18

? = D dla trybów graficznych
B dla trybów tekstowych

Zależnie od trybu i wybranej strony rejestry te zawierają wartości przesunięcia (offset) z rysunku 2. Wpisując do rejestrów nowe wartości można uzyskać "przewijanie " obrazu w pionie i poziomie bez zmian zawartości pamięci.

2.2.2. Sekwenser

Układ ten generuje sygnały sterujące pamięcią obrazu oraz takt dla cyklu wyświetlania. Koordynuje dostęp do pamięci obrazu między CPU i kontrolerem CRT. Rejestry

sekwensera przedstawiono w tabeli 3. Podobnie jak w przypadku kontrolera CRT, rejestry te (z wyjątkiem Map Mask Register) są mało przydatne w programowaniu graficznym.

Rejestr maski odwzorowania (Map Mask Register)

Służy do sterowania operacją zapisu do pamięci obrazu. Bity 0–3 odpowiadają numerom płatów pamięci. Bity 4–7 są nie używane. Wartość 1 dla dowolnego spośród bitów 0–3 oznacza odblokowanie dostępu do odpowiedniego płatu pamięci dla zapisu ze strony procesora, tzn. bajt przesyłany z CPU jest wpisywany równolegle do wszystkich odblokowanych płatów. Po inicjalizacji systemu, zawartość rejestru maski odwzorowania wynosi 0Fh.

Tabela 3

Rejestry sekwensera

Nazwa rejestru	Nr portu	Indeks
Adress	3C4	--
Reset	3C5	00
Clocking mode	3C5	01
Map Mask	3C5	02
Character Map Select	3C5	03
Memory mode	3C5	04

2.2.3. Kontroler graficzny

Układ ten umożliwia wykonywanie manipulacji na danych na drodze z pamięci obrazu do kontrolera atrybutów.

Kontroler graficzny posiada 32-bitowy rejestr danych (po 1 bajcie dla każdego płatu pamięci), który pozwala na wykonanie odczytu lub zapisu 32 bitów w 1 cyklu pamięci. Rejestr ten nosi nazwę "zatrzasku danych procesora" (Processor Latch Register). Każda operacja odczytu pamięci obrazu powoduje uaktualnienie jego zawartości, która może być następnie używana w specjalnym trybie zapisu do wykonywania 32-bitowych operacji OR,XOR lub AND z zawartością pamięci obrazu. Spis rejestrów programowalnych kontrolera graficznego wraz ze standardowymi zawartościami po inicjalizacji przedstawiono w tabeli 4. Poniższy tekst zawiera krótkie omówienie ich funkcji.

Rejestr zapisu płatów (S/R Plane Register)

Bity 0–3 odpowiadają numerom płatów VRAM.

Bity 4–7 są nie wykorzystane.

Zawartość bitów 0–3 może być wpisana do komórek pamięci odpowiednich płatów w trybie zapisu 00 (patrz Mode Register) w zależności od zawartości Enable S/R Register. Do S/R Register ładuje się z reguły numer koloru, który ma być wpisany do płatów.

Rejestr blokady zapisu (Enable S/R Register)

Bity 0–3 odpowiadają numerom blokowanych płatów.

Bity 4–7 są nie wykorzystane.

Rejestr ten służy jako maska dla rejestru omówionego wyżej. Jeśli dla danego płatu odpowiadający mu bit w Enable S/R Reg. ma wartość 1, to w czasie operacji zapisu do tego płatu, do bajtu określonego przez adres pochodzący z procesora, wpisywana jest na pozycje określone przez rejestr maski bitowej zawartość odpowiedniego bitu S/R register. W przeciwnym przypadku zawartość bitu wpisywanego do płatu pochodzi z magistrali danych procesora.

Rejestr porównywania kolorów (Color Compare Register)

Bity 0–3 – numer koloru porównywanego z zawartością ekranu w trybie odczytu 1 (patrz Mode Register).

Bity 4–7 – nie używane.

Dla trybu odczytu 1, numer koloru zapisany na bitach 0-3 jest porównywany z kolorami ośmiu punktów ekranu opisanych przez bajty czterech płatów odpowiadające podanemu z procesora adresowi. Na magistralę danych CPU wysyłany jest bajt będący wynikiem operacji. Wartość 1 na danej pozycji tego bajtu świadczy o zgodności testowanego koloru z zawartością wszystkich czterech płatów dla odpowiedniego punktu ekranu.

Rejestr wyboru operacji bitowych (Data Rotate & Function Select Register)

Rejestr ten umożliwia manipulację danymi w trybach zapisu 0 i 2.

Bity 0–2 – liczba przesunięć cyklicznych w lewo bajtu danych z procesora wykonywana przed zapisem do płatu VRAM w trybie 0

Bity 3–4 – kod operacji logicznej wykonywanej sprzętowo pomiędzy 32-bitowym zatrzaskiem danych VRAM i danymi wpisywanymi do poszczególnych płatów pamięci obrazu w trybach 0 i 2.

Wartość 00 – wpis bez modyfikacji
01 – AND
10 – OR
11 – XOR

Bity 5-7 – nie wykorzystane.

Tabela 4

Rejestry konrolera graficznego i ich zawartość po inicjalizacji systemu operacyjnego

Rejestr			Nr trybu pracy karty								
Nazwa	Port	Index	00	01	02	03	04	05	06	07	0D
Graphics 1pos.	3CC	--	00	00	00	00	00	00	00	00	00
Graphics 2pos.	3CA	--	01	01	01	00	01	01	01	01	01
Graphics Addr	3CE	--	--	--	--	--	--	--	--	--	--
Set reset	3CF	00	00	00	00	00	00	00	00	00	00
Enable S/R	3CF	01	00	00	00	00	00	00	00	00	00
Color compare	3CF	02	00	00	00	00	00	00	00	00	00
Data rotate	3CF	03	00	00	00	00	00	00	00	00	00
Read map sel.	3CF	04	00	00	00	00	00	00	00	00	00
Mode register	3CF	05	10	10	10	10	30	30	00	10	00
Miscellanous	3CF	06	0E	0E	0E	0E	0F	0F	0D	0A	00
Color no care	3CF	07	00	00	00	00	00	00	00	00	00
Bit mask	3CF	08	FF	FF	FF	FF	FF	FF	FF	FF	00

Rejestr			Nr trybu pracy							
Nazwa	Port	Index	0E	0F	10	0F	10	00	01	02	03
Graphics 1pos.	3CC	--	00	00	00	00	00	00	00	00	00
Graphics 2pos.	3CA	--	01	01	01	01	01	01	01	01	01
Set reset	3CF	--	--	--	--	--	--	--	--	--	--
Graphics Addr.	3CE	00	00	00	00	00	00	00	00	00	00
Enable S/R	3CF	01	00	00	00	00	00	00	00	00	00
Color compare	3CF	02	00	00	00	00	00	00	00	00	00
Data rotate	3CF	03	00	00	00	00	00	00	00	00	00
Read map sel.	3CF	04	00	00	00	00	00	00	00	00	00
Mode register	3CF	05	00	00	10	00	00	10	10	10	10
Miscellanous	3CF	06	05	07	07	05	05	0E	0E	0E	0E
Color no care	3CF	07	0F	0F	0F	0F	0F	00	00	00	00
Bit Mask	3CF	08	FF	FF	FF	FF	FF	FF	FF	FF	FF
						**	**	*	*	*	*

** zawartości rejestrów, gdy podłączone jest więcej niż 64 kB pamięci obrazu
* zawartości rejestrów, gdy podłączony jest monitor kolorowy wysokiej rozdzielczości

Rejestr wyboru odwzorowania odczytu (Read Map Select Register).

Bity 0–2 – decydują o wyborze płatu pamięci który jest dostępny do odczytu (w trybie odczytu 0). Dozwolonymi wartościami są 0,1,2,3.

Bity 3–7 – nie są używane.

Rejestr wyboru trybu zapisu i odczytu (Mode Register).

Bity 1 i 0 – wybór trybu zapisu:

Wartość 00 – do wszystkich płatów pamięci odblokowanych w Map Mask Register oraz w Enable S/R Register wpisywany jest bajt da-

nych pochodzący z CPU. Do płatów pamięci zablokowanych Enable S/R Reg. wpisywany jest bajt zawierający na każdej pozycji powieloną 8 razy zawartość odpowiedniego bitu z S/R Register. Wpis bajtu danych do pamięci, bez względu na jego źródło, jest zawsze poprzedzony operacjami zakodowanymi Mode Register.

Wartość 01 - tryb zapisu 1,przepisanie zawartości poszczególnych bajtów 32-bitowego zatrzasku danych (Processor Latch Register) do odpowiadających im odblokowanych w Map Mask Register płatów pamięci obrazu.

Wartość 10 - tryb zapisu 2, bity 0-3 magistrali danych procesora są wpisywane do odpowiadających im numerami płatów pod adres zadany z procesora na pozycje bitowe odblokowane w Bit Mask Register.

Wartość 11 - kombinacja nie używana.

Bit 2 - wykorzystywany przez układy karty do testów sprzętu

Bit 3 - wybór trybu odczytu:

Wartość 0 - normalny odczyt bajtu z płatu wybranego zawartością Read Map Select Register.

Wartość 1 - wykonywana jest operacja porównania koloru ośmiu punktów obrazu z numerem barwy zawartym w Color Compare Register.

Bit 4 - parzysta/nieparzysta adresacja.

Bit 5 - używany wyłącznie do emulacji trybów rozdzielczości karty CGA.

Rejestr maski koloru (Color-Don't-Care Register)

Bity 0-3 - odpowiadają płatom 0-3. Płaty, których bity są ustawione na 1, nie są brane pod uwagę przy porównywaniu zgodności kolorów w trybie odczytu 1.

Rejestr maski bitu (Bit Mask Register)

Rejestr ten jest używany jako maska we wszystkich operacjach zapisu. Jedynka na określonej pozycji rejestru odblokowuje odpowiadającą jej pozycję punktu ekranu (po jednym bicie w czterech równoległych płatach) w czasie zapisu pod zadany adres. Dzięki temu w jednej operacji zapisu można wypełniać od jednego do ośmiu punktów ekranu. Zamaskowane przez zawartość rejestru bity nie są zabezpieczone przed zapisem z 32-bitowego zatrzasku danych procesora. Dlatego bezpośrednio przed każdą operacją selektywnego wpisu koloru, należy wykonać odczyt dowolnego spośród

czterech modyfikowanych bajtów (tzn. odczytać zadany adres przy dowolnym uaktywnionym płacie pamięci).

2.2.4. Kontroler atrybutów

Układ ten steruje wyświetlaniem kolorów i atrybutów, zmianą trybów rozdzielczości oraz odwzorowaniem poszczególnych płatów pamięci obrazu na ekranie. Spis programowalnych rejestrów przedstawiono w **tabeli 5**.

Kontroler posiada 17 rejestrów opisujących dostępną na ekranie paletę barw. Przeznaczone są wyłącznie do zapisu. Ich zawartość najwygodniej jest modyfikować za pomocą funkcji 10h przerwania nr 10h BIOS.

Rejestry 0–15 definiują 16 kolorów wybranych dowolnie spośród 64. Rejestr nr 0 zawiera zawsze numer koloru tła. Ostatni z siedemnastu rejestrów opisuje barwę obramowania ekranu (ang. overscan).

Z punktu widzenia efektów graficznych, ciekawe możliwości daje rejestr o indeksie 12h (**Color Plane Enable Register**). W celu wyświetlania na ekranie barwy opisanej którymkolwiek z rejestrów palety konieczne jest podanie jego czterobitowego numeru przechowywanego w pamięci obrazu. Color Plane Enable Register pozwala na, inną niż normalna, interpretację tego numeru.

Bity 0–3 wyżej wymienionego rejestru odpowiadają numerom płatów. Zablokowanie któregokolwiek płatu wartością 0 powoduje, że zawartość tego płatu nie jest brana pod uwagę podczas wyświetlania koloru punktów. Stwarza to możliwość wykorzystania pamięci obrazu, np. do przechowywania równocześnie ośmiu monochromatycznych obrazów o rozdzielczości 640 x 350 punktów równocześnie. Bezpośredni dostęp (nie poprzez BIOS) do rejestrów kontrolera atrybutów jest utrudniony w porównaniu z poprzednimi układami.

Wybór indeksów i przesył danych w kontrolerze atrybutów odbywa się poprzez jeden adres wejścia/wyjścia 3C0h. Port ten jest przełączany we właściwy tryb (indeksów lub danych) operacjami odczytu i zapisu.

Przykładowy wpis wartości do rejestru blokady koloru płata można zapisać używając standardowych funkcji języka Turbo-C następująco:

```
/* Przełączenie portu 3C0h w tryb indeksowy */
        inportb( 0x3DA );
/* Wybór rejestru blokady koloru tła */
/* Po wykonaniu operacji zapisu ekran ulega wygaszeniu */
        outportb( 0x3C0, 0x12 );
```

```
/* Specyfikacja blokowanych płatów */
        outportb( 0x3C0, X );
/* Powtórne uaktywnienie ekranu bitem 5  */
        outportb( 0x3C0, 0x20 );
```

Tabela 5

Rejestry kontrolera atrybutów i ich zawartość po inicjalizacji

Rejestr			Tryb pracy karty												**	**	*	*	*	*
Nazwa	Port	Index	00	01	02	03	04	05	06	07	0D	0E	0F	10	0F	10	00	01	02	03
Adress	3?A	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
Palette	3C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Palette	3C0	01	01	01	01	01	13	13	17	08	01	01	08	01	08	01	01	01	01	01
Palette	3C0	02	02	02	02	02	15	15	17	08	02	02	00	00	00	02	02	02	02	02
Palette	3C0	03	03	03	03	03	17	17	17	08	03	03	00	00	00	03	03	03	03	03
Palette	3C0	04	04	04	04	04	02	02	17	08	04	04	18	04	18	04	04	04	04	04
Palette	3C0	05	05	05	05	05	04	04	17	08	05	05	18	07	18	05	05	05	05	05
Palette	3C0	06	06	06	06	06	06	06	17	08	06	06	00	00	00	06	14	14	14	14
Palette	3C0	07	07	07	07	07	07	07	17	08	07	07	00	00	00	07	07	07	07	07
Palette	3C0	08	10	10	10	10	10	10	17	10	10	10	00	00	00	38	38	38	38	38
Palette	3C0	09	11	11	11	11	11	11	17	18	11	11	00	01	08	39	39	39	39	39
Palette	3C0	0A	12	12	12	12	12	12	17	18	12	12	00	00	00	3A	3A	3A	3A	3A
Palette	3C0	0B	13	13	13	13	13	13	17	18	13	13	00	00	00	3B	3B	3B	3B	3B
Palette	3C0	0C	14	14	14	14	14	14	17	18	14	14	00	04	00	3C	3C	3C	3C	3C
Palette	3C0	0D	15	15	15	15	15	15	17	18	15	15	18	07	18	3D	3D	3D	3D	3D
Palette	3C0	0E	16	16	16	16	16	16	17	18	16	16	00	00	00	3E	3E	3E	3E	3E
Palette	3C0	0F	17	17	17	17	17	17	17	18	17	17	00	00	00	3F	3F	3F	3F	3F
Mode Control	3C0	10	08	08	08	08	01	01	01	0E	01	01	0B	0B	0B	01	08	08	08	08
Over-scan	3C0	11	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Color Plane	3C0	12	0F	0F	0F	0F	03	03	01	0F	0F	0F	05	05	05	0F	0F	0F	0F	0F
Horiz. panning	3C0	13	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
															**	**	*	*	*	*

** : zawartość rejestrów, gdy podłączone jest więcej niż 64 kB pamięci obrazu
* : zawartość rejestrów, gdy podłączony jest monitor kolorowy EGA
? = D dla trybów barwnych, B dla monochromatycznych

3. Operacje graficzne na karcie EGA

3.1. Pojedynczy zapis i odczyt VRAM - kreślenie punktu

Plastrowa organizacja pamięci obrazu karty EGA w trybie wysokiej rozdzielczości utrudnia dostęp ze strony procesora. Odczyt koloru bądź nakreślenie punktu na ekranie wymaga nie tylko wykonania operacji odczytu lub zapisu odpowiedniego adresu pamięci, ale także zaprogramowania rejestrów wewnętrznych karty. Wykorzystując dostępne tryby zapisu (patrz par.1.2) operację kreślenia punktu można przedstawić w postaci następujących procedur w języku C.

```
/*
   tablice cząstkowych offsetów inicjowane funkcją vmode()
   przyspieszają obliczanie adresu efektywnego
   pixela w VRAM
                                                          */
   unsigned Yaddr[350] ;
   unsigned Xaddr[640] ;

/*
   tablica zawartości rejestru BitMask
   inicjowana funcją vmode
                                                          */
   unsigned char Bitmask[640] ;

/*
   wskaźnik do początku pamięci obrazu
                                                          */
   char far * EGA_ScreenPtr= (char far * )0xA0000000L ;

/*
   funkcja inicjująca tryb graficzny
   wykorzystuje przerwanie 0x10 do przełączania
   trybu, inicjuje tablicę Bitmask[]
   i tablice offsetów
                                                          */
   void  vmode(int mode)
   {
      union REGS reg ;
      int i;

      for (i=0;i<=349;i++)
         Yaddr[i] = 80 * i;
      for (i=0;i<=639;i++)
      {
         Xaddr[i]= i>>3 ;
         Bitmask[i]= 128 >> (i&7) ;
      };

      reg.h.ah=0 ;/* wywol. przerw. 0x10 BIOS-a  */
      reg.h.al=mode;
      int86(0x10,&reg,&reg);
   }
```

```
/*
   funkcja kreśląca punkt -- metoda 1
   parametry we ; x,y    -- współrzędne punktu
                color -- nr koloru
   tryb zapisu 0, wartość color wpisywana jest do S/R reg.
   a stamtąd do VRAM
                                                          */
   void  setpixel1(int x,int y,int color)
   {
      unsigned ofs;

      outp( 0x3CE, 0 ); /* wybór S/R register */
      outp( 0x3CF, color ); /* S/R register = color */
      outp( 0x3CE, 1 ); /* wybor Enable S/R register */
      outp( 0x3CF, 0x0F);/* blokada zapisu danych z CPU */
      outp( 0x3CE, 8 ); /* wybor Bit-Mask reg. */
      outp( 0x3CF, Bitmask[x] );/* ładowanie maski */

      ofs = Xaddr[x] + Yaddr[y] ;

      /* odczyt/zapis pamięci obrazu */
      /* odczyt ładuje Processor Latch zabezpieczając przed */
      /* zapisem  z jego strony bity zamaskowane w BitMask reg.*/
      EGA_ScreenPtr[ofs] = EGA_ScreenPtr[ofs];

      /* przywrócenie rejestrom wartosci poczatkowych */
      outp( 0x3CF , 0xFF ); /* Bit-Mask. register = 0xFF */
      outp( 0x3CE , 1 ); /* wybor Enable S/R reg. */
      outp( 0x3CF , 0 ); /* Enable S/R reg. = 0 */
   }

/*
   funcja kreśląca punkt -- metoda 2
   parametry we ; x,y    -- współrzędne punktu
                color -- nr koloru
   tryb zapisu 0, wartość color wpisywana jest
   do Map Mask register, ze strony CPU następuje zapis
   bajtu jedynek
                                                          */
   void  setpixel2(int x,int y,int color)
   {
      unsigned ofs;
      char temp;

      outp( 0x3C4 , 2 );/* wybór Map Mask register */
      outp( 0x3C5 , color ); /* Map Mask register = color */

      outp( 0x3CE , 8 ); /* wybór Bit-Mask reg. */
      outp( 0x3CF , Bitmask[x] );/* ładowanie maski */

      ofs = Xaddr[x] + Yaddr[y] ;

      temp= EGA_ScreenPtr[ofs];
      EGA_ScreenPtr[ofs] = 0xFF;

      /* przywrócenie rejestrom wartosci poczatkowych */
      outp( 0x3CF , 0xFF ); /* Bit-Mask register = 0xFF */
      outp( 0x3C5 , 0x0F ); /*Mask Map reg.= 0x0F */
   }
```

```
/*
   funcja kreśląca punkt -- metoda 3
   parametry we ; x,y   -- współrzędne punktu
                color -- nr koloru
   tryb zapisu 2, każdy z 4 młodszych bitów wartości
   color przesyłanej na mag. danych CPU wpisywany jest
   do odpowiedniego płatu pamięci obrazu
*/
void  setpixel3(int x,int y,int color)
{
   unsigned ofs;
   char temp;

   outp( 0x3CE , 5 );/*wybór Mode register */
   outp( 0x3CF , 2 ); /* tryb zapisu 2 */

   outp( 0x3CE , 8 ); /* wybór Bit-Mask reg. */
   outp( 0x3CF , Bitmask[x] );/* ładowanie maski */

   ofs = Xaddr[x] + Yaddr[y] ;

   temp = EGA_ScreenPtr[ofs];
   EGA_ScreenPtr[ofs] = color;

   /* przywrócenie rejestrom wartśsci początkowych */
   outp( 0x3CF , 0xFF ); /* Bit-Mask register = 0xFF */
   outp( 0x3CE , 5 );/*wybór Mode register */
   outp( 0x3CF , 0 ); /* Mode register= 0 */
}
```

Odczyt jawnego numeru pojedynczego punktu jest dosyć uciążliwy z uwagi na rozrzucenie informacji dotyczącej 1 pixela w 4 różnych bajtach VRAM. Na skompletowanie 4-bitowego numeru potrzebne są 4 operacje odczytu pamięci w trybie 0 oraz 4 operacje zapisu do Read Map Select register, kolejno czytane płaty.

W wielu procedurach graficznych zamiast odczytu wartości koloru punktu wystarczy sprawdzenie, czy kolor ten jest równy określonej liczbie (np. przy wypełnianiu konturów przez spójność). Poniższy wydruk przedstawia realizację takiej operacji przy użyciu trybu odczytu nr 1.

```
/*
   testowanie koloru punktu ekranu -- odczytu w trybie 1
   we : x,y,color - wspolrzedne punktu i zadany kolor
   wy : zwracana wartosc 1--tak, 0--nie
*/
int ispixelcolor(int x , int y , int color)
{
   int YES ;

   outp( 0x3CE, 2 );/* wybor Color Compare reg. */
   outp( 0x3CF, color );

   outp( 0x3CE, 5 );/* wybor Mode register */
   outp( 0x3CF, 0x08 );/* wybor trybu odczytu 1 */
```

```
    YES= EGA_ScreenPtr[ Xaddr[x] + Yaddr[y] ] & Bitmask[x];

    outp( 0x3CF, 0 );/* poprzednia wartosci do Mode register */
    return(YES);
}
```

Często odczyt jest potrzebny jedynie w celu skopiowania obrazu w inne miejsce. Do przesyłów typu VRAM ---> VRAM najwygodniejsze jest wykorzystanie 32-bitowego rejestru Procesor Latch. Przy dowolnej operacji odczytu w trybie 0 rejestr ten jest ładowany zawartością 4 bajtów pamięci obrazu, po 1 bajcie z każdego płatu, co odpowiada poziomej kresce złożonej z 8 punktów obrazu. Zawartość Processor latch może być przepisana pod dowolny inny adres VRAM, z możliwością maskowania zawartości (można w ten sposób skopiować od 1 do 8 punktów równocześnie). Najwygodniejszy do przepisywania jest tryb zapisu 1. Poniższa sekwencja instrukcji kopiuje punkt o współrzędnych x1,y1 w miejsce x+deltax,y+deltay, gdzie deltax jest całkowitą wielokrotnością 8.

```
/* wybór trybu  zapisu 1*/
outportb(0x3CE,5);
outportb(0x3CF,1);

offset1= Xaddr[x] + Yaddr[y];
offset2= Xaddr[x+deltax] + Yaddr[y+deltay];

/* Maskowanie sąsiednich punktów */
/* wokół miejsca przeznaczenia przed zapisem */
outportb(0x3CE,8);
outportb(0x3CF,Bitmask[x+deltax]);

/* operacja odczytu i zapisu */
/* sygnał odczytu na magistrali CPU */
/* powoduje ładowanie Processor Latch */
/* sygnał zapisu powoduje wykonanie operacji */
/* zapisu w trybie 1 */
EGA_ScreenPtr[ofs2] = EGA_ScreenPtr[ofs1];

/* przywrócenie rejestrom poprzedniej wartości*/
outportb(0x3CF,0xFF);
outportb(0x3CE,5);
outportb(0x3CF,1);
```

3.2. Szybkie kreślenie linii poziomych

Poszczególne płaty pamięci obrazu odwzorowują punkty ekranu w kolejności od lewej do prawej strony, z góry w dół. W związku z tym dla każdej poziomej linii ekranu punkty o współrzędnej spełniającej warunek $i*8 <= X <= i*8 +7$, gdzie i - liczba całkowita z przedziału $<0,79>$, są przechowywane w tym samym bajcie każdej mapy bitowej, pod tym samym adresem w każdym z płatów. Punty te mogą zostać zapełnione przy użyciu jednej operacji dostępu do VRAM w dowolnym trybie zapisu. Korzystając z tej

własności można zaproponować szybki algorytm kreślenia jednobarwnych poziomych odcinków, który przyspieszy w przybliżeniu 8-krotnie wykonywanie takich operacji graficznych, jak np. wypełnianie konturów metodą kontroli parzystości. Przedstawiona poniżej procedura dzieli poziomy odcinek na 3 części: środkową, kreśloną operacjami zapisu po 8 punktów na raz, oraz 2 części boczne - lewą i prawą, stanowiące "resztę" z obcięcia końców odcinka wartościami współrzędnej X podzielnymi przez 8. Każdy z "końców" zawiera mniej niż 8 punktów i jest kreślony 1 operacją zapisu przy użyciu od odpowiedniej zawartości rejestru BitMask.

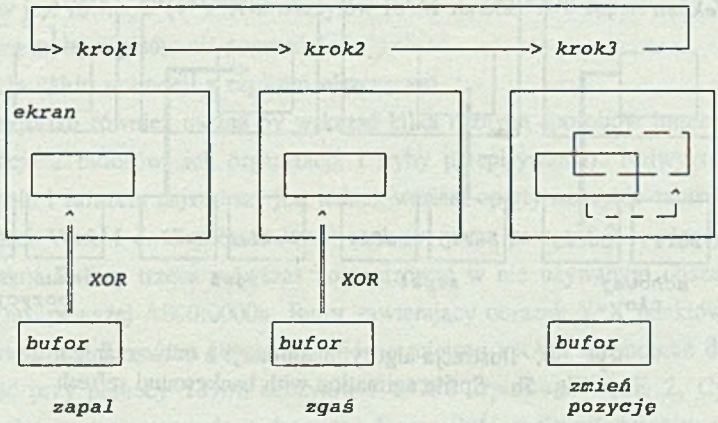
```
/*
   Szybkie kreslenie poziomego odcinka na karcie EGA
   w trybie wys. rozdzielczości
   wykorzystano tryb zapisu do VRAM nr 0
   Parametry we :
      xl,xr -- współrzędne X końców odcinka
         y  -- współrzędna Y poziomej linii
      color -- nr koloru 0..15
*/
void line_horizontal(int xl,int xr,int y,int color)
{
   int xll,xrr,l,r;
   unsigned ofs;
   unsigned char maskleft,maskright;

   xll = xl - (xl & 7);
   xrr = xr - (xr & 7);
   maskleft = 0xFF >> (xl & 7);
   maskright = 0xFF << (xrr + 7 - xr);

   outportb(0x3CE,1);
   outportb(0x3CF,0x0F);/* blokada zapisu z CPU */

   outportb(0x3CE,0);
   outportb(0x3CF,color);/* S/R Reg. = color */

   ofs = Xaddr[xl] + Yaddr[y];
   outportb(0x3CE,8);/* wybór BitMask Reg. */
   if(xll==xrr)
   {
      /* przypadek, gdy nie ma części środkowej */
      outportb(0x3CF,maskleft&maskright);
      EGA_ScreenPtr[ofs++] = EGA_ScreenPtr[ofs];
   }
   else
   {
      /* kreślenie lewego końca */
      outportb(0x3CF,maskleft);/* ładowanie maski lewego końca */
      EGA_ScreenPtr[ofs++] = EGA_ScreenPtr[ofs];

      outportb(0x3CF,0xFF);/* kasowanie maski bitowej */
      xll+=8;
      /* pętla kreśląca środek odcinka */
      while(xll<xrr)
      {
         EGA_ScreenPtr[ofs++]=0;
         xll += 8;
      }
      /* kreślenie prawego końca */
```

```
        outportb(0x3CF,maskright);/* maska prawego końca*/
        EGA_ScreenPtr[ofs] = EGA_ScreenPtr[ofs];
    }

    /* przywrócenie rejestrom poprzednich wartości */
    outportb(0x3CF,0xFF);
    outportb(0x3CE,1);
    outportb(0x3CE,0);
}
```

3.3. Przesuwanie fragmentu obrazu po ekranie

Wrażenie ruchu na monitorze rastrowym można uzyskać stosując 1 z 2 zasadniczych technik:

1) wielokrotne kreślenie obrazu animowanego obiektu w coraz to nowych miejscach ekranu (kształt rysunku może być zmienny, generacja i wyliczanie parametrów kreślonego kształtu następuje na bieżąco dla każdego kadru),
2) generacja obrazu do bufora (buforów) i wielokrotne kopiowanie przygotowanego w ten sposób wzorca (wzorców) w różne miejsca pamięci obrazu.



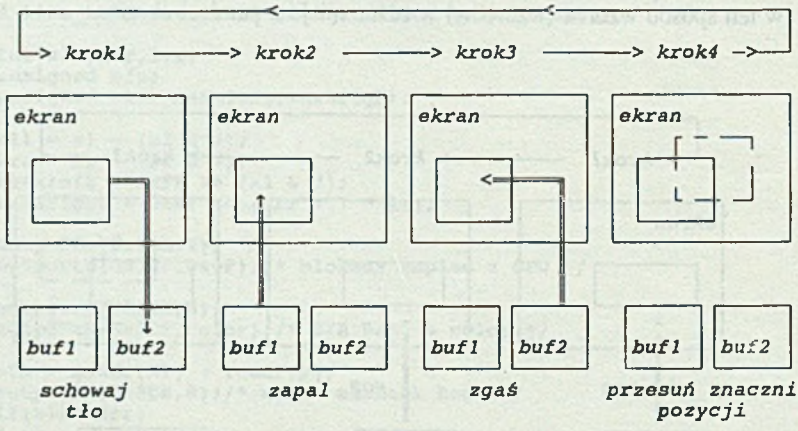
Rys. 5a. Ilustracja algorytmu animacji z operacją XOR
Fig. 5a. Sprite animation with XOR

Sposób 1 ze względu na dużą złożoność obliczeniową algorytmów generacji poszczególnych kadrów oraz trajektorii ruchu jest właściwy dla końcówek graficznych dużych komputerów lub "stacji roboczych" (ang. workstations).

Sposób 2 cechuje duża ilość przesyłów, kadry są wyliczane "a priori", a następnie przesyłane punkt po punkcie. Szybkość animacji zależy więc nie od złożoności rysunku, lecz od jego rozmiaru. W przypadku terminala graficznego połączonego z CPU łączem szeregowym o niewielkiej prędkości transmisji animacja tym sposobem napotyka na poważne ograniczenia (w celu uzyskania wrażenia płynności konieczne jest "przepchnięcie" przez łącze minimum 25 kadrów na sekundę).

W programach graficznych na mikrokomputerach z uwagi na stosunkowo szybką komunikację z pamięcią obrazu końcówki graficznej (z reguły VRAM dostępna jest w przestrzeni adresowej CPU), często imituje się ruch w sposób 2. Ze względu na sposób "nakładania" wzorca na ekran można wyróżnić 2 warianty cyklu animacji:

1) kopiowanie wzorca z operacją XOR (rys.5a),
2) kopiowanie niszczące z odtwarzaniem tła (rys.5b).



Rys. 5b. Ilustracja algorytmuanimacji z odtwarzaniem tla
Fig. 5b. Sprite animation with background refresh

Implementacja powyższych algorytmów dla karty EGA sprowadza się do lokalizacji buforów (w pamięci operacyjnej lub w nie wyświetlanym aktualnie obszarze pamięci obrazu), doboru organizacji danych (sposobu odwzorowania obrazu) wewnątrz buforów oraz trybów zapisu i odczytu.

ad 1) Realizacja cyklu animacji z operacją XOR

Wariant 1:

- bufor wzorca umieszczony w pamięci operacyjnej CPU,

- odwzorowanie obrazka 2 pixele na 1 bajt,
- zapis do VRAM punkt po punkcie przy w trybie 0 lub 2 z zaprogramowaną operacją XOR w Mode Register,
- możliwość kopiowania w dowolne miejsce ekranu (brak ograniczeń na krok zmiany pozycji),
- liczba operacji we/wy do pamięci w 1 cyklu animacji dla wzorca o rozmiarze X*Y punktów jest równa 2*(Y*X/2 odczytów RAM + Y*X zapisów do VRAM).

Wariant 2:

- bufor w RAM w przestrzeni adresowej CPU,
- organizacja bufora analogiczna do organizacji VRAM, tzn. obraz przechowywany jako 4 mapy bitowe wzorca, w tym przypadku jedna po drugiej,
- zapis do pojedynczych płatów VRAM poszczególnych map bitowych bufora (zapis całymi bajtami w trybie 0, Bit Mask Register odblokowany), przy zastosowaniu Map Mask Register,
- szerokość wzorca oraz współrzędne poziomu lewego brzegu obszaru na ekranie, w który wpisywany jest wzorzec, muszą być wielokrotnością 8,
- liczba operacji we/wy do pamięci w 1 cyklu animacji dla wzorca o rozmiarze X*Y punktów jest równa 2*(4*Y*X/8 odczytów RAM + 4*Y*X/8 zapisów do VRAM) o połowę mniej zapisów niż poprzednio.

ad 2) Realizacja cyklu animacji z zapisem niszczącym

W tym przypadku również można by wskazać kilka różnych sposobów implementacji (różne lokalizacje 2 buforów, ich organizacja i tryby przepisywania). Najwygodniejszy w programowaniu i zarazem najszybszy jest jednak wariant oparty na trybie zapisu 2 i odczycie zawartości VRAM do Processor Latch register (patrz paragraf 2.1, algorytm kopiowania punktu). Bufory trzeba wówczas zorganizować w nie używanym obszarze pamięci obrazu, np. powyżej A800:0000h. Bufor zawierający obrazek Y*X punktów, gdzie X jest podzielne przez 8, można skopiować w inne miejsce VRAM lub pobrać do niego nową zawartość przy pomocy Y*X/8 odczytów i Y*X/8 zapisów w trybie 2. Cykl animacji zawiera 3 tego typu operacje, wobec tego łączna ilość zapisów/odczytów w cyklu wynosi 3/8*Y*X odczytów VRAM + 3/8*Y*X zapisów VRAM. Jedyną poważną wadą takiego rozwiązania jest skokowość ruchu - minimalne przesunięcie wynosi 8 punktów.

3.4. Przewijanie ekranu, przechowywanie i podgląd obrazu 800x600

Przy pełnej (256kB) konfiguracji pamięci karta EGA jest standardowo wykorzystywana do przechowywania 2 obrazów (stron 640x350). Zmianę wyświetlanej strony zape-

wnia odpowiednia procedura BIOS (przerwanie nr 10h). Rezygnując ze skokowego przełączania stron i wykorzystując bezpośrednio rejestry Start Adress High i Start Adress Low kontrolera CRT można uzyskać szereg ciekawych efektów opartych na płynnym przesuwaniu wyświetlanego na monitorze obrazu. W najprostszym przypadku zawartość VRAM może być traktowana jako ciągły obraz, złożony maksymalnie z 819 poziomych linii po 640 punktów, "podglądany" przez "okno" 640x350 przesuwane w górę i w dół dzięki dekrementacji i inkrementacji offsetu przechowywanego w ww. rejestrach o wartość n*80, gdzie n - liczba poziomych linii. Nieco bardziej złożony jest podgląd obrazu o rozdzielczości poziomej większej niż 640 i przewijanie "w lewo" i "w prawo". Zamieszczony poniżej fragment programu realizuje to zadanie dla obrazu 800x600.

```
/* wskaźnik do początku aktualnie wyświetlanego obszaru */
char far * EGA_ScreenPtr= (char far * )0xA0000000L ;

/* Wskaźnik do początku pamięci obrazu */
char far * EGA_vram= (char far * )0xA0000000L ;

/* Wskaźnik do obszaru "środkowego słupa obrazu 800x600" */
char far * VRAM_Bar=(char far * )0xA0000000L ;

/* Wskaźnik do obszaru obrazu spoza "środkowego słupa"*/
char far * Rest= (char far * )0xA000BF68L; /*offset +49000L */

/* Wskaźnik do obszaru zawierającego pomocniczy bufor wymiany */
char far * Buf= (char far * )0xA000EE48L; /*offset +61000L*/

/* struktura określająca położenie lewego górnego rogu        */
/* ekranu 640x350 względem lewego górnego rogu obrazu 800x600 */
struct { int x; int y; } Pos= {0,0};

/*
   Funkcja kreśląca punkt na obrazie 800x600
*/
   void set_pixel_Hires(int x, int y, int color)
   {
      unsigned ofs,mask;
      if(x<Pos.x)
         ofs= 49000L + (x>>3) +y*20 ;
      else
         if(x >= Pos.x+640)
            ofs= 49000L + (long)((x - Pos.x - 640)>>3)+y*20;
         else
            ofs= (x>>3) +(long)y*80 ;

      mask= 128 >> (x&7);
      outp(0x3CE,0);outp(0x3CF,color);
      outp(0x3CE,1);outp(0x3CF,0x0F);
      outp(0x3CE,8);outp(0x3CF,mask);

      EGA_vram[ofs] = EGA_vram[ofs];

      outp(0x3CF,0xFF);
      outp(0x3CE,1);outp(0x3CF,0);
```

```
/*
   Funkcja inkrementująca 16-bitowy offset w rejestrach
   Start Adress High i Start Adress Low wartość deltaofs
                                                          */
void scroll(int deltaofs)
{
   unsigned OFS;
   unsigned char;

   outportb(0x3D4,0x0C);
   OFS = inportb(0x3D5) << 8;
   outportb(0x3D4,0x0D);
   OFS += inportb(0x3D5);
   OFS += deltaofs;
   outportb(0x3D5, OFS & 0xFF);
   outportb(0x3D4,0x0C);
   outportb(0x3D5, OFS>>8 );
}

/*
   Funkcja realizująca przesunięcie w prawo o 8 punktów
   "okna" 640x350 po obrazie 800x600.
                                                          */
void move_screen_right(void)
{
   unsigned ofs,i,j;
   outp(0x3CE,5);outp(0x3CF,1);

   for(i=0,j=0 ;i<600 ; i++,j+=80)
      Buf[i] = VRAM_Bar[j] ;

   scroll(+1);EGA_ScreenPtr++ ;VRAM_Bar++;

   for(j= Pos.x>>3,i=79 ; i<47999L ; j+=20,i+=80)
      VRAM_Bar[i] = Rest[j] ;

   for(i= Pos.x>>3,j=0 ; j<600 ; j++,i+=20 )
      Rest[i] = Buf[j] ;

   Pos.x += 8;

   outp(0x3CE,5);outp(0x3CF,0);
}

/*
   Funkcja realizująca przesunięcie w lewo o 8 punktów
   "okna" 640x350 po obrazie 800x600.
                                                          */
void move_screen_left(void)
{
   unsigned ofs,i,j;
   outp(0x3CE,5);outp(0x3CF,1);

   for(i=0,j=79 ; i<600 ; i++,j+=80)
      Buf[i] = VRAM_Bar[j] ;

   scroll(-1); EGA_ScreenPtr-- ;VRAM_Bar--;

   for(j= (Pos.x-1)>>3,i=0 ; i<47920L ; j+=20,i+=80)
      VRAM_Bar[i] = Rest[j] ;

   for(i= (Pos.x-1)>>3,j=0 ; j<600 ; j++,i+=20 )
      Rest[i] = Buf[j] ;

   Pos.x -= 8;
```

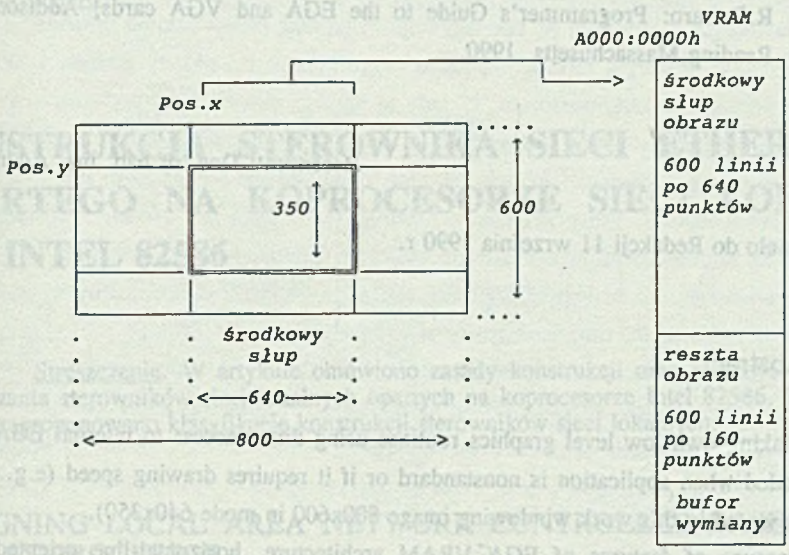
```
    outp(0x3CE,5);outp(0x3CF,0);
  }

/*
  Funkcja realizująca przesunięcie w dół o n  linii
  "okna" 640x350 po obrazie 800x600.
*/
void move_screen_down(int n)
{
   scroll(n*80);
   Pos.y+=n;
   EGA_ScreenPtr+=n*80;
}

/*
  Funkcja realizująca przesunięcie w górę  o n linii
  "okna" 640x350 po obrazie 800x600.
*/
void move_screen_up(int n)
{
   scroll(-n*80);
   Pos.y-=n;
   EGA_ScreenPtr-=n*80;
}
```

Przestrzeń adresowa pamięci obrazu została w tym przypadku podzielona na 3 obszary: obszar "środkowego słupa" przechowujący 600 poziomych linii po 640 punktów, obszar "reszty" - 600 linii po 160 punktów oraz bufor 600 adresów wykorzystywany wymianie danych między pierwszym i drugim obszarem (rys.6). Wyświetlany ekran 640x350 znajduje się zawsze w obrębie obszaru pierwszego. Przesunięcia "w górę" i "w dół" są realizowane poprzez inkrementację lub dekrementację opisywanych poprzednio rejestrów kontrolera CRT o wielokrotność 80 bajtów. Przesuwy "w lewo" i "w prawo" wymagają, oprócz wpisu odpowiednich wartości do rejestrów, wymiany danych między obszarem 1 i 2. W programie wykorzystano do tego celu tryb zapisu 1.

4. Podsumowanie

Bezpośrednie programowanie rejestrów sterownika EGA jest, jak wykazują przedstawione przykłady, czynnością dosyć żmudną i wymagającą dobrej znajomości architektury karty. Z doświadczeń autora wynika, że postępowanie takie jest uzasadnione w przypadkach, gdy wymagana jest duża szybkość kreślenia, oszczędność pamięci lub niestandardowe wykorzystanie ekranu (np. opisane wyświetlanie obrazu 800x600).

Ze względu na specyficzną strukturę pamięci obrazu i dostępne tryby zapisu i odczytu VRAM, algorytmy kreślenia jednobarwnych pełnych figur bądź przesuwania fragmentów obrazu, powinny być ukierunkowane na kreślenie lub kopiowanie poziomych odcinków.



Rys. 6. Odwzorowanie obrazu 800x600 w pamięci VRAM
Fig. 6. Mapping 800x600 image data into display memory

Niniejszy artykuł prezentuje sposoby realizacji efektywnych elementarnych procedur graficznych oraz przykłady niestandardowych trickowych efektów. Zamieszczone w tekście procedury opracowano i uruchomiono dla wygody w języku "C" (kompilator Turbo-C 2.0).

LITERATURA

[1] IBM Update Technical Reference Manual.

[2] COMPAQ Enhanced Color Graphics Board, Technical Reference Guide, COMPAQ Computer Corporation, Dezember 1986.

[3] D.Meiners: Regie uber register. Programmierung der Original IBM EGA Karte, c't Magazin 12/87.

[4] Aleksander Schulze: Bessere Grafik mit PCs, c't Magazin 5/86.

[5] G.Sutta i S.Blair: Programmer's Guide to the EGA/VGA, Brady Books, New York 1988.

[6] R.Ferraro: Programmer's Guide to the EGA and VGA cards, Addison-Wesley, Reading,Massachusetts 1990.

Recenzent: Doc. dr hab. inż. Adam Mrózek

Wpłynęło do Redakcji 11 września 1990 r.

Abstract

Making own low level graphics routines using direct access to internal EGA registers is needed when application is nonstandard or if it requires drawing speed (e.g. animation or presented in this work windowing image 800x600 in mode 640x350).

Because of features of EGA VRAM architecture, horizontal-line oriented graphics algorithms ought to be preferred.

This work presents examples of effective implementation of low level graphics routines and few non standard, graphic tricks for EGA card in mode 10h.

All included functions are compiled and running using TURBO-C 2.0 compiler.