Laboratorium Architektury Komputerów

Ćwiczenie 6

Obsługa przerwań sprzętowych

Wprowadzenie

Współczesne, wielozadaniowe systemy operacyjne starają się izolować programy użytkowe od sprzętu komputerowego. Programy mogą się komunikować z urządzeniami wyłącznie za pośrednictwem funkcji oferowanych przez system operacyjny. W tej sytuacji przeprowadzenie jakichkolwiek eksperymentów ilustrujących zasady sterowania urządzeniami komputerowymi nie może być zrealizowane na poziomie zwykłej aplikacji. Powyższe uwagi odnoszą się także do mechanizmów obsługi przerwań sprzętowych, które są tematem niniejszego ćwiczenia.

W ramach niniejszego ćwiczenia spróbujemy zrealizować przedstawione zamierzenia wykorzystując środowisko systemu DOS. System ten już od wielu lat nie jest używany, ale jego funkcje zostały przejęte przez system MS Windows, który jeszcze do niedawna oferował możliwość realizacji programów przeznaczonych do wykonywania w środowisku systemu DOS.

System operacyjny DOS powstał w początkowym okresie rozwoju komputerów PC i z założenia był systemem jednozadaniowym, przystosowanym do pracy przy dość ubogich zasobach sprzętowych (np. pamięć operacyjna 640 KB, procesor z zegarem 8 MHz). System DOS nie posiada żadnych mechanizmów ochrony systemu operacyjnego i pozwala w szczególności na ingerencję (oczywiście w przemyślany sposób) w mechanizmy obsługi przerwań. Stanowi to podstawę do ilustracji mechanizmów obsługi przerwań w ramach niniejszego ćwiczenia.

Przypomnijmy, że procesory zgodne z architekturą x86 mogą pracować w dwóch trybach pracy:

— *rzeczywistym*, który naśladuje i pewnym stopniu rozszerza funkcje procesora 8086, który stanowił pierwowzór rodziny *x86*;

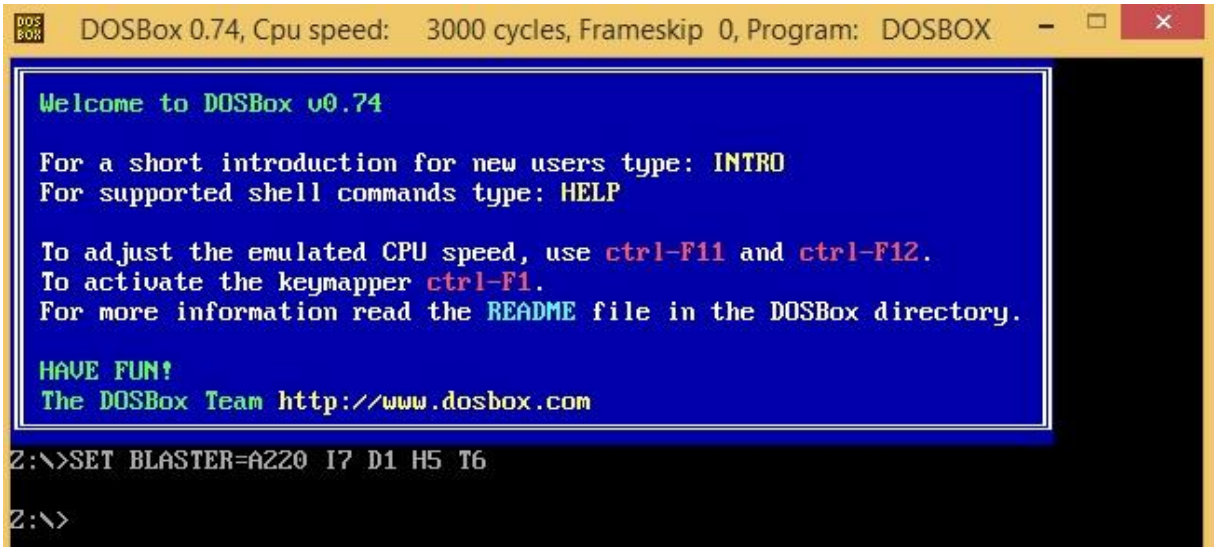
— *chronionym*, w którym dostępne są mechanizmy wielozadaniowości i stosowane są odmienne niż w trybie rzeczywistym sposoby adresowania i ochrony.

W ramach trybu chronionego wprowadzono także (począwszy od procesora 80386, r. 1985) specjalny podtryb, określany jako tryb V86 (tryb wirtualny 8086), w którym, z punktu widzenia wykonywanych programów, procesor działa prawie dokładnie tak samo jak w trybie rzeczywistym. Tryb V86 używany był do wykonywania programów "DOSowych" w środowisku systemu Windows. Tryb V86 nie jest jednak dostępny w najnowszych, 64-bitowych wersjach systemu MS Windows. W tej sytuacji programy przewidziane do wykonywania w środowisku systemu DOS można uruchamiać jedynie za pomocą maszyny wirtualnej, np. DOSBox, która opisana jest poniżej.

Maszyna wirtualna DOSBox

W laboratoriach komputerowych MKZL, w celu uruchomienia maszyny wirtualnej *DOSBox* należy kliknąć dwukrotnie na ikonę umieszczoną na pulpicie. Ponadto, najnowsza wersja instalacyjna tego programu w postaci pliku `DOSBox0.74-3-win32-installer.exe` dostępna jest na wielu stronach internetowych.

Po uruchomieniu maszyny wirtualnej na ekranie pojawi się pokazane niżej okno. Do asemblacji programów będziemy używać asemblera `masm`, a do konsolidacji programu `link` (wersja 16-bitowa). Programy te, udostępnione na serwerze `\\nmkzl1\Public\AKO\Cw_6`, należy przegrać do katalogu użytkownika na dysku **d:**, np. `d:\Zofia`. W tym katalogu należy także umieścić pliki źródłowe programów w asemblerze.



Dla podanej lokalizacji wskazane jest utworzenie dysku wirtualnego skojarzonego z ww. katalogiem. W tym celu w okienku konsoli maszyny wirtualnej trzeba wprowadzić polecenie:

```
mount  d  d:\Zofia
```

Następnie, w oknie konsoli maszyny wirtualnej należy zmienić bieżący napęd dyskowy (wirtualny) na `d:` Teraz można przeprowadzić asemblację i linkowanie programu przykładowego:

```
masm  gwiazdki.asm,,,;
link  gwiazdki.obj;
```

Uwaga: powyższe polecenia asemblacji i linkowania muszą być zakończone znakiem średnika.

W rezultacie powstanie plik `gwiazdki.exe`, który można uruchomić w okienku maszyny wirtualnej (wpisać nazwę programu `gwiazdki` i nacisnąć klawisz Enter). Program można zakończyć poprzez naciśnięcie klawisza X.

W wielu przypadkach dokonujemy zmiany zawartości katalogu skojarzonego z dyskiem wirtualnym na zewnątrz maszyny wirtualnej. W celu odświeżenia dysku wirtualnego wystarczy nacisnąć kombinację klasy Ctrl F4.

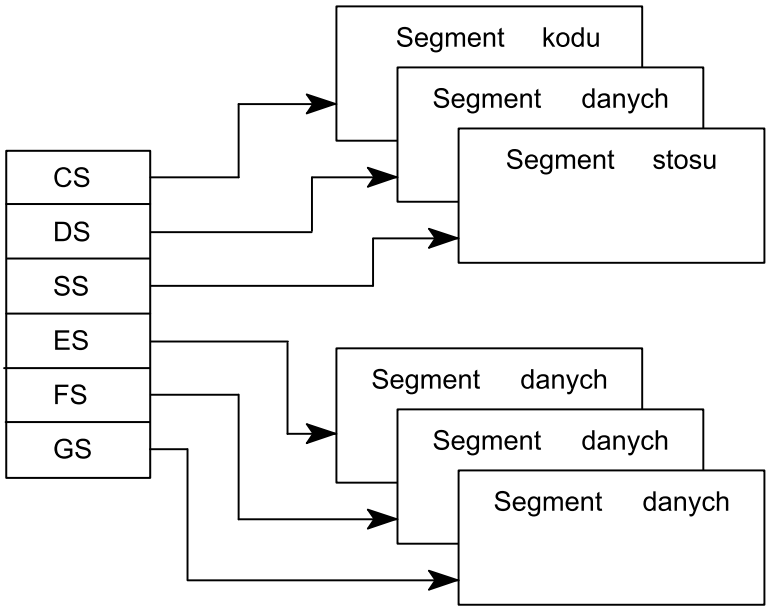
Adresowanie pamięci w trybie rzeczywistym

W opisanych dalej programach przykładowych przyjęto, że będą one wykonywane w *trybie rzeczywistym*. Wprawdzie nie będzie to prawdziwy tryb rzeczywisty, ale jego symulacja realizowana za pomocą opisanej wyżej maszyny wirtualnej. Zrozumienie zasad działania tych programów wymaga poznania mechanizmu obsługi przerwań sprzętowych, jak również znajomości schematu adresowania pamięci w trybie rzeczywistym.

W trybie rzeczywistym przyjęto zasady adresowania pamięci właściwe dla architektury segmentowej. W pamięci operacyjnej komputera, w części przeznaczonej na programy użytkowe wyróżnia się trzy obszary: obszar przeznaczony na rozkazy programu, obszar przeznaczony na dane (statyczne) i obszar stosu (dane dynamiczne). W trybie rzeczywistym procesorów rodziny *x86* adresy początkowe tych obszarów wskazują 16-bitowe rejestry segmentowe (zob. rysunek):

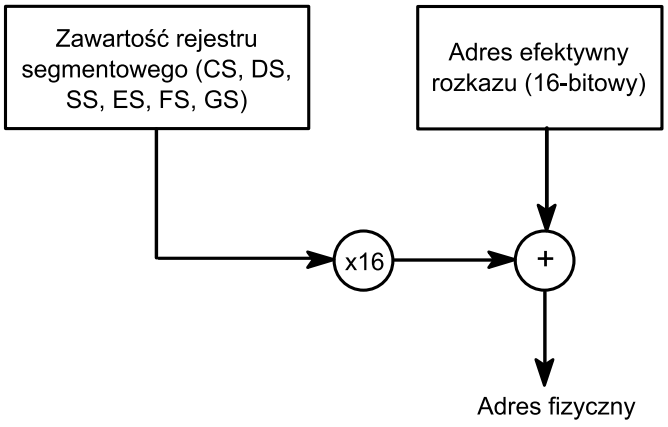
- rejestr CS wskazuje adres początkowy części rozkazowej programu,
- rejestr DS wskazuje adres początkowy obszaru danych programu,
- rejestr SS wskazuje obszar stosu.

Używane są też trzy inne rejestry segmentowe, które pełnią funkcje pomocnicze.



Rola podanych tu rejestrów segmentowych we współczesnych systemach operacyjnych, które działają z reguły w trybie chronionym, została zmarginalizowana – stosowane są inne techniki adresowania.

Wymienione rejestry segmentowe odgrywają natomiast istotną rolę w operacjach adresowania w trybie rzeczywistym. W obliczeniach adresów zawartości rejestrów segmentowych są zawsze mnożone przez 16. Ponieważ największa 16-bitowa liczba binarna bez znaku (NKB) ma wartość 65535, więc zawartość każdego z tych rejestrów może wskazywać adres o wartości maksymalnej 16 * 65535 = 1048560. Oznacza to, że za pomocą rejestrów segmentowych można określać położenie obszarów danych i rozkazów w pamięci operacyjnej o rozmiarze 1 MB.



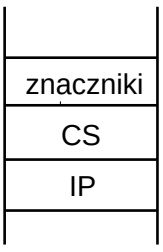
W omawianym mechanizmie adresowania przyjęto, że liczba umieszczona w polu adresowym rozkazu określa odległość danej, liczoną w bajtach, od początku obszaru danych (który wskazuje rejestr DS). Analogiczne reguły przyjęto dla obszaru rozkazów i stosu. W rezultacie adres fizyczny danej stanowi sumę pomnożonej przez 16 zawartości rejestru DS i odległości podanej w polu adresowym rozkazu. Algorytm obliczania adresu fizycznego pokazany jest na rysunku.

W praktyce programowania (w trybie rzeczywistym) zazwyczaj nie podaje się 20-bitowego adresu fizycznego rozkazu (instrukcji) lub danej, ale adres ten wyraża się w postaci dwóch liczb szesnastkowych rozdzielonych dwukropkiem: pierwsza z tych liczb wskazuje początek segmentu, w którym znajduje omawiany obiekt, druga zaś określa przesunięcie wewnątrz segmentu, tj. odległość obiektu od początku segmentu (wyrażona w bajtach). Przykładowo, 32-bitowy programowy licznik czasu BIOSu umieszczony jest w lokacji pamięci o adresie 40H:6CH, tzn. w lokacji pamięci o adresie fizycznym 40H × 16 + 6CH = 46CH. W odniesieniu do podanego schematu adresowani mówimy, że adres lokacji pamięci został wyrażony w postaci *segment : offset*.

Obsługa przerwań sprzętowych

Przerwania sprzętowe są pewnymi zdarzeniami zachodzącymi w urządzeniach komputera, które wymagają podjęcia niezwłocznej obsługi. W takim przypadku urządzenie wysyła do procesora sygnał przerwania, który powoduje, że procesor przerywa wykonywanie bieżącego programu i rozpoczyna wykonywanie innego programu, zazwyczaj stanowiącego część systemu operacyjnego. Zadaniem tego programu jest zbadanie przyczyn nadejścia sygnału przerwania i podjęcie odpowiedniej akcji, np. poinformowanie użytkownika, że kopiowanie pliku z Internetu zostało zakończone, albo też że operacja drukowania została zatrzymana ze względu na brak papieru. Po wykonaniu wszystkich czynności związanych z obsługą przerwania system operacyjny wznawia wykonywanie przerwanego programu.

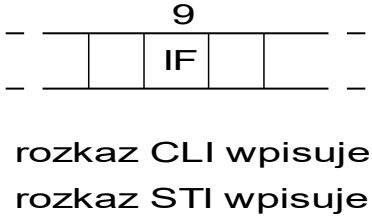
Tak więc procesor, oprócz wykonywania rozkazów programu, musi być przygotowany do obsługi przerwań, które pojawiają się asynchronicznie. Zazwyczaj procesor podejmuje obsługę przerwania po zakończeniu aktualnie wykonywanego rozkazu. Następnie zapisuje położenie w pamięci (adres) kolejnego rozkazu, który zostałby wykonany, gdyby nie nadeszło przerwanie. Położenie to, określane jako *ślad* przerwania, zapisywane jest na stosie (w trybie rzeczywistym elementy stosu są 16-bitowe). Rysunek po lewej stronie przedstawia strukturę śladu przerwania w trybie rzeczywistym (architektura x86). Pola (16-bitowe) CS i IP (ang. instruction pointer – wskaźnik instrukcji) zawierają opis położenia w pamięci rozkazu, który zostanie wykonany po zakończeniu obsługi przerwania. W tym momencie, na podstawie pola "znaczniki" zostanie odtworzona też zawartość 16-bitowego



rejestru znaczników (FLAGS).

Po zakończeniu obsługi przerwania musi nastąpić wznowienie wykonywania programu głównego. Obsługa przerwanie nie może mieć żadnego wpływu na wykonywanie programu głównego, w szczególności nie mogą nastąpić jakiekolwiek zmiany zawartości rejestrów i znaczników. Ponieważ rejestry i znaczniki będą używane w trakcie obsługi przerwania, trzeba je więc od razu zapamiętać i odtworzyć bezpośrednio przez zakończeniem obsługi. Działania te wykonywane są zazwyczaj programowo, z częściowym wspomaganiem sprzętowym. Przykładowo, w procesorach zgodnych z architekturą x86 automatycznie zapamiętywany jest tylko rejestr znaczników FLAGS (lub EFLAGS w trybie 32-bitowym), inne rejestry muszą być zapamiętane przez program obsługi.

W procesorach zgodnych z architekturą x86 warunkiem przyjęcia przerwania sprzętowego (generowanego przez urządzenie zewnętrzne) jest stan znacznika IF = 1. Znacznik IF (ang. interrupt flag) w rejestrze znaczników (bit nr 9) określa zezwolenie na przyjmowanie przerwań: procesor może przyjmować przerwania tylko wówczas, gdy IF=1. Znacznik IF jest automatycznie zerowany w chwili przyjęcia przerwania.



Możliwe jest zablokowanie przyjmowania przerwań poprzez wyzerowanie znacznika IF. W programie, do zmiany stanu znacznika IF można zastosować rozkazy CLI (IF ← 0) lub STI (IF ← 1).

Obsługa przerwań jest ściśle związana z tablicą wektorów przerwań (w trybie chronionym tablica ta nosi nazwę *tablicy deskryptorów przerwań*). W trybie rzeczywistym tablica wektorów przerwań zawiera 256 adresów, z których każdy zajmuje 4 bajty. Adresy kodowane są w postaci *segment:offset*. Tablica umieszczona jest w pamięci począwszy od adresu fizycznego 0 (aczkolwiek jej położenie może zostać zmienione).

Po zapisaniu śladu na stosie procesor odszukuje w tablicy wektorów przerwań adres procedury obsługi przerwania i rozpoczyna ją wykonywać. Numer wektora przerwania, w którym zawarty jest adres procedury obsługi zależy w ustalony sposób od numeru linii IRQ, poprzez którą nadszedł sygnał przerwania. W przypadku programów 16-bitowych wykonywanych w systemie Windows/DOS numer wektora stanowi powiększony o 8 numer linii IRQ (dla linii IRQ 8 ÷ IRQ 15 numer powiększany jest o 104).

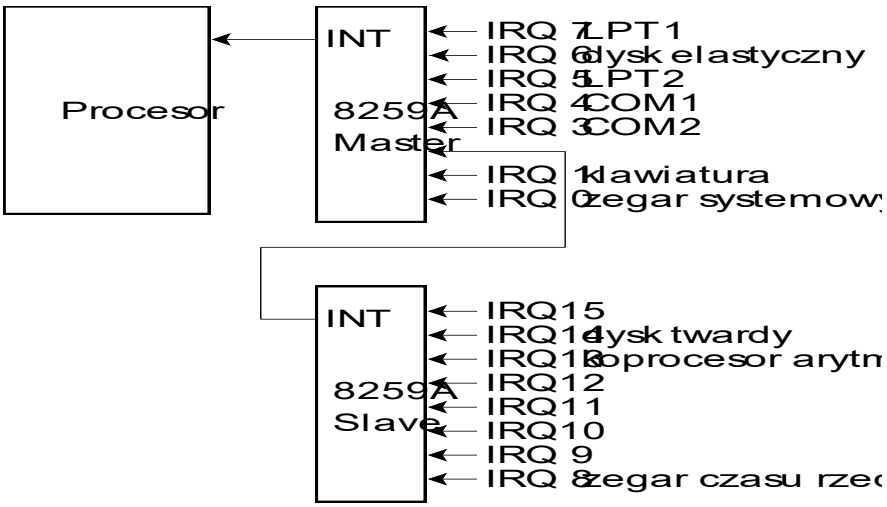
Podprogram obsługi przerwania kończy rozkaz IRET, która powoduje wznowienie wykonywania przerwanego programu poprzez odtworzenie rejestrów IP, CS i FLAGS, na podstawie śladu zapamiętanego na stosie.

Sterownik przerwań

Zazwyczaj każde urządzenie dołączone do komputera jest w stanie generować sygnały przerwań. Wymaga to odpowiedniego zorganizowania systemu przerwań, tak poszczególne przerwania były przyjmowane wg ustalonej hierarchii. Na ogół procesor nie jest przygotowany do bezpośredniej obsługi przerwań, zwłaszcza jeśli jest zainstalowanych dużo urządzeń. Stosowane są różne systemy obsługi przerwań; niekiedy zainstalowana jest wspólna linia przerwań dla wszystkich urządzeń — po nadejściu przerwania procesor sprawdza stany

poszczególnych urządzeń identyfikując urządzenie, które wysłało przerwanie (metoda odpytywania). W innych systemach linia przerwań przechodzi przez wszystkie zainstalowane urządzenia ustawione wg priorytetów.

Aktualnie w komputerach PC system przerwań obsługiwany jest przez układ APIC (dawniej używano dwóch układów typu 8259), który pełni rolę "sekretarki" procesora. W trybie rzeczywistym procesora układ APIC pracuje w trybie konwencjonalnym naśladując pracę swoich poprzedników. Możemy zatem odnieść nasze rozważania do układów 8259, co pozwoli na dokładniejsze wyjaśnienie techniki obsługi przerwań. Układy te, pracujące w konfiguracji kaskadowej, mogą obsługiwać do 15 źródeł przerwań. Sygnały przerwań z poszczególnych urządzenia kierowane są do układów 8259 poprzez linie oznaczone symbolami IRQ 0 – IRQ 15.



Z każdą linią IRQ (ang. interrupt request) skojarzony jest wektor przerwania w tablicy wektorów (deskryptorów) przerwań. Skojarzenie to wykonywane poprzez odpowiednie zaprogramowanie układu 8259 — wykonuje to system operacyjny podczas inicjalizacji. Typowe przyporządkowanie stosowane w systemie DOS podane jest poniższej tabeli.

IRQ	Urządzenie	Nr wek-tora
0	zegar systemowy, przerwanie wysyłane przez układ 8254 (w systemie DOS około 18 razy/s)	8
1	klawiatura, przerwanie wysyłane po naciśnięciu lub zwolnieniu klawisza	9
2	połączone z drugim układem 8259	
3	łącze szeregowe COM2	11
4	łącze szeregowe COM1	12
5	łącze równoległe LPT2	13
6	sterownik dyskietek	14
7	łącze równoległe LPT1	15

IRQ	Urządzenie	Nr wek-tora
8	zegar czasu rzeczywistego, przerwanie generowane o ustalonym czasie (budzenie)	112
9		113
10		114
11		115
12		116
13	koprocesor arytmetyczny	117
14	sterownik dysku twardego	118
15		119

Przykładowo, nadejście sygnału IRQ 1 powoduje przerwanie i uruchomienie podprogramu obsługi przerwania, którego adres znajduje się w wektorze nr 9 (obsługa klawiatury).

Komunikacja z urządzeniami zewnętrznymi

Urządzenia zewnętrzne komputera są zazwyczaj podłączone poprzez rożnego typu układy pośredniczące, nazywane niekiedy adapterami. Układy pośredniczące dopasowują standardy sygnałowe procesora i płyty głównej do specyficznych wymagań poszczególnych urządzeń. Kilkanaście lat temu układy pośredniczące miały postać kart rozszerzeniowych instalowanych na płycie głównej komputera. Współcześnie funkcje realizowane przez te karty zostały zintegrowane w postaci układu na płycie głównej nazywanego *chipsetem*. Karty rozszerzeniowe mogą być nadal instalowane, aczkolwiek stosowane są w bardziej rozbudowanych konfiguracjach komputerów.

W takim ujęciu procesor nie steruje urządzeniami bezpośrednio, ale wykonuje to za pośrednictwem układów wejścia/wyjścia. W układach wejścia/wyjścia istotną rolę odgrywają zazwyczaj cztery rejestry:

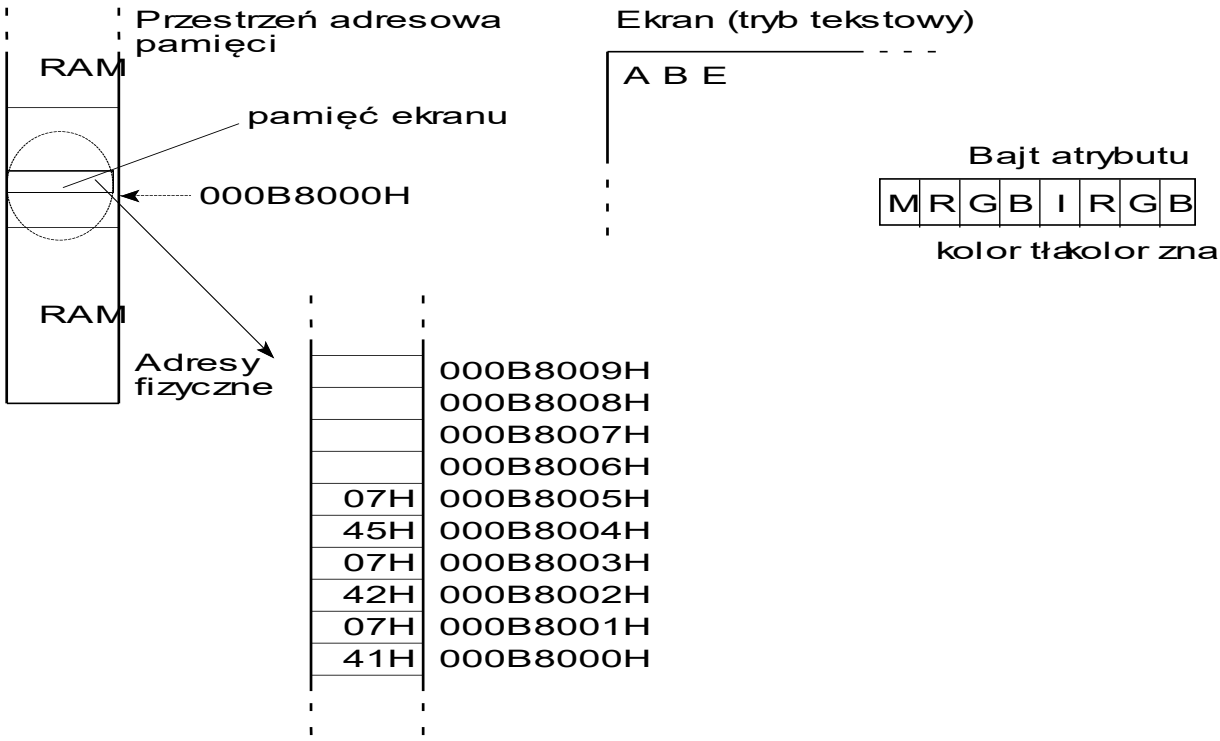
- rejestr stanu zawiera informacje o stanie urządzenia, w szczególności pozwala stwierdzić czy urządzenie jest zajęte, czy dane są gotowe do odczytania lub czy wystąpił błąd;
- rejestr sterujący przyjmuje polecenia, które urządzenie ma wykonać;
- rejestr danych wysyłanych do urządzenia;
- rejestr danych odebranych z urządzenia.

Stosowane są dwie metody dostępu do zawartości rejestrów układów wejścia/wyjścia:

- rejestry udostępniane są jako zwykłe komórki pamięci w przestrzeni adresowej pamięci – mówimy wówczas o *współadresowalnych układach wejścia/wyjścia*;
- rejestry urządzenia dostępne są w odrębnej przestrzeni adresowej zwanej *przestrzenią adresową wejścia-wyjścia* lub *przestrzenią adresową portów* – takie rozwiązanie określane jest czasami jako *izolowane wejście-wyjście.*

Współadresowalne układy wejścia/wyjścia

Typowym przykładem wykorzystania techniki układów współadresowalnych jest pamięć ekranu w komputerach PC. W trybie tekstowym sterownika graficznego znaki wyświetlane na ekranie stanowią odwzorowanie zawartości obszaru pamięci od adresu fizycznego B8000H. Pamięć ta należy do przestrzeni adresowej procesora, ale zainstalowana jest na karcie sterownika graficznego.

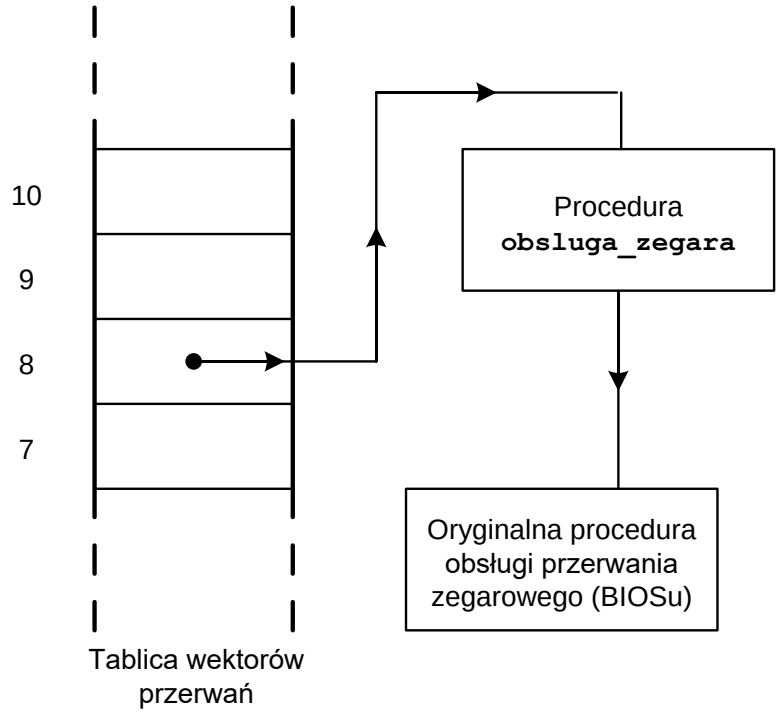


Każdy znak wyświetlany na ekranie jest opisywany przez dwa bajty w pamięci ekranu: bajt o adresie parzystym zawiera kod ASCII znaku, natomiast następny bajt zawiera opis sposobu wyświetlania, nazywany *atrybutem znaku*. Kolejne bajty omawianego obszaru odwzorowywane są w znaki na ekranie począwszy od pierwszego wiersza od lewej do prawej, potem drugiego wiersza, itd. tak jak przy czytaniu zwykłego tekstu.

Przykład obsługi przerwania zegarowego

W celu zilustrowania mechanizmu przerwań podano dwa przykłady obsługi przerwania zegarowego, które w systemie Windows/DOS generowane jest co ok. 55 ms. Pierwszy program przykładowy (`gwiazdki.asm`) pracuje w trybie tekstowym i wyświetla znaki * w takt przerwań zegarowych. Naciśnięcie klawisza X powoduje zakończenie wykonywania programu. Drugi program przykładowy (`linie.asm`) pracuje w trybie graficznym i wyświetla pionowe linie w różnych kolorach — w tym przypadku zakończenie wykonywania programu następuje po naciśnięciu dowolnego klawisza.

Podana dalej procedura (podprogram) obsługi przerwania zegarowego `obsluga_zegara` uruchamiana jest regularnie co ok. 55 ms po nadejściu każdego przerwania zegarowego. Przy każdym wywołaniu tej procedury do pamięci ekranu wysyłany jest kod ASCII znaku gwiazdki * (bajty o adresach parzystych), a na sąsiednie bajty (adresy nieparzyste) wprowadzane są kody koloru znaku (tu: biały na czarnym tle). Bieżąca pozycja w pamięci ekranu między kolejnymi wywołaniami procedury obsługi przerwania przechowywana jest w zmiennej `licznik`. Na końcu tej procedury umieszczony jest rozkaz skoku bezwarunkowego `jmp dword PTR cs:wektor8`, który przekazuje sterowanie do oryginalnej procedury obsługi przerwania zegarowego (wchodzącej w skład systemu BIOS). Ilustruje to poniższy rysunek.



Warto zwrócić uwagę na sposób instalacji procedury (podprogramu) `obsluga_zegara`. Standardowo, wektor nr 8 zawiera adres procedury obsługi przerwania zegarowego, wchodzącej w skład systemu BIOS. W trakcie instalacji procedury wektor ten zostaje zapamiętany w zmiennej `wektor8`, a do wektora nr 8 w *tablicy wektorów przerwań* zostaje wpisany adres procedury `obsluga_zegara`. W trakcie wpisywania nowego adresu do wektora blokowane są przerwania (rozkaz `cli`) — trzeba bowiem brać pod uwagę sytuację, w której przerwanie zegarowe nadeszłoby bezpośrednio po zapisaniu pierwszej części adresu (rozkaz `mov ds:[32], bx`). W takiej sytuacji połowa adresu byłaby już zmieniona, a połowa niezmieniona, więc cały adres w wektorze nr 8 wskazywałby przypadkową lokację pamięci.

Na końcu procedury obsługi przerwania `obsluga_zegara` nie jest wykonywany zwykły rozkaz powrotu z podprogramu `RET` (lub `IRET`), lecz wykonywany jest skok do oryginalnej procedury BIOSu (rozkaz `jmp dword PTR cs:wektor8`). Rozkaz `jmp` nie korzysta ze śladu pozostawionego na stosie, lecz z adresu oryginalnej procedury, który został wcześniej zapamiętany w zmiennej `wektor8`.

```
; Program gwiazdki.asm
; Wyświetlanie znaków * w takt przerwań zegarowych
; Uruchomienie w trybie rzeczywistym procesora x86
; lub na maszynie wirtualnej
; zakończenie programu po naciśnięciu klawisza 'x'
; asemblacja (MASM 4.0):     masm  gwiazdki.asm,,,;
; konsolidacja (LINK 3.60):  link  gwiazdki.obj;

.386
rozkazy   SEGMENT   use16
          ASSUME    CS:rozkazy
```

```
;=================================================================
; procedura obsługi przerwania zegarowego

obsluga_zegara      PROC

; przechowanie używanych rejestrów
     push      ax
     push      bx
     push      es

; wpisanie adresu pamięci ekranu do rejestru ES - pamięć
; ekranu dla trybu tekstowego zaczyna się od adresu B8000H,
; jednak do rejestru ES wpisujemy wartość B800H,
; bo w trakcie obliczenia adresu procesor każdorazowo mnoży
; zawartość rejestru ES przez 16
     mov       ax, 0B800h     ;adres pamięci ekranu
     mov       es, ax

; zmienna 'licznik' zawiera adres bieżący w pamięci ekranu
     mov       bx, cs:licznik

; przesłanie do pamięci ekranu kodu ASCII wyświetlanego znaku
; i kodu koloru: biały na czarnym tle (do następnego bajtu)
     mov       byte PTR es:[bx], '*'         ; kod ASCII
     mov       byte PTR es:[bx+1], 00000111B ; kolor

; zwiększenie o 2 adresu bieżącego w pamięci ekranu
     add       bx,2

; sprawdzenie czy adres bieżący osiągnął koniec pamięci ekranu
     cmp       bx,4000
     jb        wysw_dalej     ; skok gdy nie koniec ekranu

; wyzerowanie adresu bieżącego, gdy cały ekran zapisany
     mov       bx, 0

;zapisanie adresu bieżącego do zmiennej 'licznik'
wysw_dalej:
     mov       cs:licznik,bx

; odtworzenie rejestrów
     pop       es
     pop       bx
     pop       ax

; skok do oryginalnej procedury obsługi przerwania zegarowego
     jmp       dword PTR cs:wektor8

; dane programu ze względu na specyfikę obsługi przerwań
; umieszczone są w segmencie kodu
licznik        dw   320  ; wyświetlanie począwszy od 2. wiersza
```

```
wektor8        dd   ?

obsluga_zegara          ENDP

;=========================================================
; program główny - instalacja i deinstalacja procedury
; obsługi przerwań

; ustalenie strony nr 0 dla trybu tekstowego
zacznij:
     mov       al, 0
     mov       ah, 5
     int       10

     mov       ax, 0
     mov       ds,ax          ; zerowanie rejestru DS


; odczytanie zawartości wektora nr 8 i zapisanie go
; w zmiennej 'wektor8' (wektor nr 8 zajmuje w pamięci 4 bajty
; począwszy od adresu fizycznego 8 * 4 = 32)
     mov       eax,ds:[32]    ; adres fizyczny 0*16 + 32 = 32
     mov       cs:wektor8, eax

; wpisanie do wektora nr 8 adresu procedury 'obsluga_zegara'
     mov       ax, SEG obsluga_zegara ; część segmentowa adresu
     mov       bx, OFFSET obsluga_zegara    ; offset adresu

     cli       ; zablokowanie przerwań

; zapisanie adresu procedury do wektora nr 8
     mov       ds:[32], bx   ; OFFSET
     mov       ds:[34], ax   ; cz. segmentowa

     sti         ;odblokowanie przerwań


; oczekiwanie na naciśnięcie klawisza 'x'
aktywne_oczekiwanie:
     mov ah,1
     int 16H
; funkcja INT 16H (AH=1) BIOSu ustawia ZF=1 jeśli
; naciśnięto jakiś klawisz
     jz        aktywne_oczekiwanie

; odczytanie kodu ASCII naciśniętego klawisza (INT 16H, AH=0)
; do rejestru AL
     mov       ah, 0
     int       16H
     cmp       al, 'x'        ; porównanie z kodem litery 'x'
     jne       aktywne_oczekiwanie     ; skok, gdy inny znak
```

```
; deinstalacja procedury obsługi przerwania zegarowego

; odtworzenie oryginalnej zawartości wektora nr 8
     mov       eax, cs:wektor8
     cli
     mov       ds:[32], eax   ; przesłanie wartości oryginalnej
                              ; do wektora 8 w tablicy wektorów
                              ; przerwań
     sti

; zakończenie programu
     mov       al, 0
     mov       ah, 4CH
     int       21H
rozkazy        ENDS

nasz_stos      SEGMENT   stack
     db        128 dup (?)
nasz_stos      ENDS

END  zacznij
```

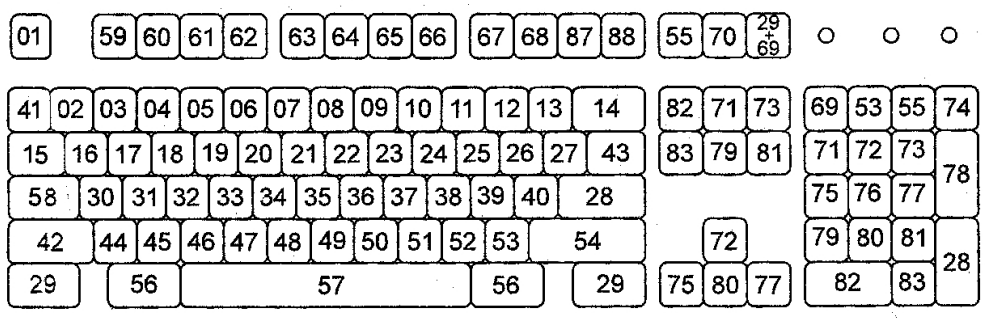
Zadanie 6.1. Zmodyfikować podany wyżej program w taki sposób, by gwiazdki wyświetlane były w kolorze żółtym na niebieskim tle.

Zadanie 6.2. Zmodyfikować podany wyżej program w taki sposób, by kolejne gwiazdki pojawiały się na ekranie co około 1 s. Wskazówka: w systemie DOS przerwania zegarowe wysyłane są co ok. 55 ms.

Zadanie 6.3. Zmodyfikować podany wyżej program w taki sposób, by gwiazdki wyświetlane były w kolejnych liniach pionowych.

Przykład obsługi przerwania z klawiatury

Sterownik klawiatury stosowany w komputerach PC stanowi przykład układu wejścia-wyjścia sterowanego przez specjalizowane procesory. Każdemu przyciskowi klawiatury przyporządkowano ustalony kod 8-bitowy, nazywany *kodem pozycji* albo numerem klawisza (ang. scan code). Kody pozycji typowej klawiatury pokazane są na rysunku.



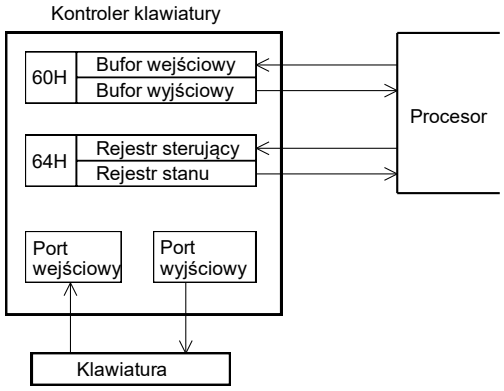
Jednolity system numeracji obejmuje zarówno zwykłe klawisze znakowe jak też wszystkie klawisze sterujące i funkcyjne (np. Shift, Ctrl, ...) – pewne wyjątki od tej zasady omawiane są dalej. Po naciśnięciu lub zwolnieniu dowolnego klawisza mikrokontroler (mikroprocesor) klawiatury formuje *kod naciśnięcia* (ang. make code) lub kod zwolnienia (ang. break code) klawisza — kod ten zostaje przesłany do układów płyty głównej komputera. Kod zwolnienia zawiera kod naciśnięcia poprzedzony bajtem F0H.

Kody naciśnięcia i zwolnienia jednoznacznie określają klawisze, ale nie są identyczne z kodami pozycji. Kody naciśnięcia i zwolnienia nie występują na poziomie programowania, przykładowe kody dla kilku klawiszy podaje poniższa tabelka:

Klawisz	kod pozycji	kod naciśn.	kod zwoln.
A	30	1CH	F0H 1CH
B	48	32H	F0H 32H
C	46	21H	F0H 21H
Q	17	15H	F0H 15H

Układ na płycie głównej komputera (mikrokontroler, np. 8042 lub 8741 lub 8742) odtwarza oryginalny kod pozycji (w przypadku zwolnienia klawisza: kod pozycji + 128). Niektóre klawisze (np. Ctrl lewy i prawy) mają jednakowy kod pozycji — w celu odróżnienia ich, przed kodem pozycji jednego z klawiszy przesyłany jest dodatkowy bajt E0H lub E1H. Odtworzony kod pozycji (lub kod pozycji zwiększony o 128) zostaje udostępniony w porcie 60H – kod pozycji naciśniętego (lub zwolnionego) klawisza można więc odczytać za pomocą rozkazu `in al, 60H` . Jednocześnie generowany jest sygnał przerwania poprzez linię IRQ 1, z którą skojarzony jest wektor przerwania nr 9.

Zarówno mikrokontroler na płycie głównej jak też mikrokontroler klawiatury mogą być programowane za pomocą rozkazów wysyłanych przez główny procesor poprzez porty 60H i 64H. Istnieje kilkanaście rozkazów, które mogą być przesyłane do klawiatury, m.in. w ten sposób można określić parametry tzw. autorepetycji, czyli samoczynnego powtarzania wysyłania kodu odpowiadającego przytrzymywanemu dłużej klawiszowi. Schemat blokowy sterownika klawiatury zainstalowanego na płycie głównej pokazany jest na rysunku.



Zadanie 6.4.

Wzorując się na podanym wcześniej przykładzie obsługi przerwania zegarowego napisać program, który przejmie obsługę przerwań z klawiatury (wektor 9). Podprogram wywołany wskutek wystąpienia przerwania z klawiatury powinien wyświetlić na ekranie (w postaci dziesiętnej) kod pozycji naciśniętego (lub zwolnionego) klawisza. Wyświetlenie

liczby zrealizować poprzez bezpośredni zapis do pamięci ekranu (wykorzystać podprogram „wyswietl_AL”).

```
; podprogram  'wyswietl_AL' wyświetla zawartość rejestru AL
; w postaci liczby dziesiętnej bez znaku
wyswietl_AL  PROC
; wyświetlanie zawartości rejestru AL na ekranie wg adresu
; podanego w ES:BX
; stosowany jest bezpośredni zapis do pamięci ekranu

; przechowanie rejestrów
     push ax
     push cx
     push dx

     mov  cl, 10    ; dzielnik

     mov  ah, 0     ; zerowanie starszej części dzielnej

; dzielenie liczby w AX przez liczbę w CL, iloraz w AL,
; reszta w AH (tu: dzielenie przez 10)
     div  cl

     add    ah, 30H ; zamiana na kod ASCII
     mov    es:[bx+4], ah ; cyfra jedności

     mov    ah, 0
     div    cl ; drugie dzielenie przez 10
     add    ah, 30H ; zamiana na kod ASCII
     mov    es:[bx+2], ah ; cyfra dziesiątek
     add    al, 30H ; zamiana na kod ASCII
     mov    es:[bx+0], al ; cyfra setek

; wpisanie kodu koloru (intensywny biały) do pamięci ekranu
     mov    al, 00001111B
     mov    es:[bx+1],al
     mov    es:[bx+3],al
     mov    es:[bx+5],al

; odtworzenie rejestrów
     pop    dx
     pop    cx
     pop    ax
     ret       ; wyjście z podprogramu
wyswietl_AL  ENDP
```

Przykład wyświetlania linii w trybie graficznym

Poniżej podano kod drugiego programu przykładowego, który wyświetla na ekranie pionowe linie w różnych kolorach. W odróżnieniu od poprzedniego, program ten wykorzystuje tryb graficzny (200 wierszy, w każdym wierszu 320 pikseli).

```
; Program linie.asm
; Wyświetlanie znaków * w takt przerwań zegarowych
; Uruchomienie w trybie rzeczywistym procesora x86
; lub na maszynie wirtualnej
; zakończenie programu po naciśnięciu dowolnego klawisza
; asemblacja (MASM 4.0):     masm  gwiazdki.asm,,,;
; konsolidacja (LINK 3.60):  link  gwiazdki.obj;

.386
rozkazy        SEGMENT        use16
          ASSUME         cs:rozkazy

linia     PROC

; przechowanie rejestrów
     push ax
     push bx
     push es

     mov  ax, 0A000H     ; adres pamięci ekranu dla trybu 13H
     mov  es, ax

     mov  bx, cs:adres_piksela     ; adres bieżący piksela
     mov  al, cs:kolor
     mov  es:[bx], al ; wpisanie kodu koloru do pamięci ekranu

; przejście do następnego wiersza na ekranie
     add  bx, 320

; sprawdzenie czy cała linia wykreślona
     cmp  bx, 320*200
     jb   dalej     ; skok, gdy linia jeszcze nie wykreślona

; kreślenie linii zostało zakończone - następna linia będzie
; kreślona w innym kolorze o 10 pikseli dalej
     add  word PTR cs:przyrost, 10
     mov  bx, 10
     add  bx, cs:przyrost
     inc  cs:kolor  ; kolejny kod koloru

; zapisanie adresu bieżącego piksela
dalej:
     mov  cs:adres_piksela, bx
```

```
; odtworzenie rejestrów
        pop     es
        pop     bx
        pop     ax

; skok do oryginalnego podprogramu obsługi przerwania
; zegarowego
        jmp     dword PTR cs:wektor8


; zmienne procedury
kolor          db   1         ; bieżący numer koloru
adres_piksela  dw   10        ; bieżący adres piksela
przyrost  dw   0
wektor8        dd   ?

linia ENDP

; INT 10H, funkcja nr 0 ustawia tryb sterownika graficznego
zacznij:
     mov  ah, 0
     mov  al, 13H   ; nr trybu
     int  10H

     mov  bx, 0
     mov  es, bx         ; zerowanie rejestru ES
     mov  eax, es:[32]   ; odczytanie wektora nr 8
     mov  cs:wektor8, eax; zapamiętanie wektora nr 8

; adres procedury 'linia' w postaci segment:offset
     mov  ax, SEG linia
     mov  bx, OFFSET linia

     cli       ; zablokowanie przerwań

; zapisanie adresu procedury 'linia' do wektora nr 8
     mov  es:[32], bx
     mov  es:[32+2], ax

     sti                 ; odblokowanie przerwań

czekaj:
     mov  ah, 1          ; sprawdzenie czy jest jakiś znak
     int  16h            ; w buforze klawiatury
     jz   czekaj

     mov  ah, 0     ; funkcja nr 0 ustawia tryb sterownika
     mov  al, 3H    ; nr trybu
     int  10H

; odtworzenie oryginalnej zawartości wektora nr 8
```

```
     mov  eax, cs:wektor8
     mov  es:[32], eax

; zakończenie wykonywania programu
     mov  ax, 4C00H
     int  21H
rozkazy   ENDS

stosik    SEGMENT stack
     db   256 dup (?)
stosik    ENDS

END  zacznij
```

Zadanie 6.5. Zmodyfikować podany wyżej program w taki sposób, by na ekranie zostały wyświetlone przekątne (z pewnym przybliżeniem).