ROZDZIAŁ DWUDZIESTY**: KLAWIATURA PC**

Klawiatura PC jest podstawowym ludzkim urządzeniem wejściowym w systemie. Chociaż wydaje się raczej przyziemna, klawiatura jest podstawowym urządzeniem wejściowym dla większości programów, więc nauczenie się jak oprogramować właściwie klawiaturę jest bardzo ważne w rozwijaniu aplikacji.
IBM i niezliczone wytwórnie klawiatur produkują liczne klawiatury dla PC i kompatybilnych. Większość nowoczesnych klawiatur dostarcza przynajmniej 101 różnych klawiszy i są umiarkowanie kompatybilne z 101 Klawiszową rozszerzoną Klawiaturą IBM PC / AT. Te które dostarczają dodatkowych klawiszy ogólnie programują te klawisze do emitowania sekwencji uderzeń klawiszy lub pozwalają użytkownikowi oprogramować sekwencję uderzeń w klawisze dodatkowych klawiszy. Ponieważ 101 klawiszowa klawiatura jest wszechobecna, zakładamy, że używamy jej w tym rozdziale.

Kiedy IBM rozwijał pierwszy PC, używali bardzo prostego sprzęgu pomiędzy klawiaturą a komputerem. Kiedy IBM wprowadził PC/AT, zupełnie przeprojektowali interfejs klawiatury. Od czasu wprowadzenia PC/AT, prawie każda klawiatura jest dostosowana do standardu PC/AT. Nawet kiedy IBM wprowadził system PS/2, zmiany w interfejsie klawiatury i zgodna oddolnie z projektem PC/AT. Dlatego też, rozdział ten będzie również ograniczony do urządzeń kompatybilnych z PC/AT ponieważ tylko kilka systemów i klawiatur PC/XT jest jeszcze w użyciu.

Jest pięć głównych elementów klawiatury jakie będziemy rozpatrywać w tym rozdziale – podstawowe informacje o klawiaturze, interfejs DOS, interfejs BIOS, podprogram obsługi przerwania klawiatury int 9 i sprzętowy interfejs do klawiatury. Ostatnia sekcja tego rozdziału bezie omawiała jak udawać wejście klawiatury w naszych aplikacjach.

20.1 PODSTAWY KLAWIATURY

Klawiatura PC jest komputerem systemowym w swoim własnym rozumieniu. Ukryto wewnątrz klawiatury chip mikrokontrolera 8042, który stale skanuje przełączniki klawiatury aby sprawdzić czy naciśnięto jakiś klawisz. Ten proces pracuje równolegle z normalną działalnością PC, dlatego klawiatura nigdy nie gubi naciśniętego klawisza jeśli 80x86 w PC jest zajęty.

Typowo uderzenie w klawiaturę zaczyna się kiedy użytkownik naciśnie klawisz na klawiaturę. To zamyka elektryczny kontakt w przełączniku mikrokontroler i wskazuje ,że naciśnięto przełącznik. Niestety , przełączniki (będące mechanicznymi rzeczami) nie zawsze zamykają (mają kontakt) tak gładko. Często, kontakty odbijają się kilka razy zanim przejdą do stałego kontaktu. Jeśli chip mikrokontrolera odczyta stały przełącznik, to odbijanie się kontaktów będzie wyglądało jak bardzo szybka seria naciśnięć klawisza i zwolnień. To może generować wielokrotne naciśnięcia klawiszy na głównym komputerze, zjawisko znane jako odbijanie klawisza, powszechne w wielu tanich i starych klawiaturach. Ale nawet na droższych i nowszych klawiaturach odbijanie klucza jest problemem jeśli popatrzymy na przełączniki milion razy na sekundę; mechaniczne przełączniki nie mogą po prostu uspokoić się tak szybko. Dlatego wiele algorytmów skanowania klawiatury kontroluje jak często skanowana jest klawiatura. Typowy niedrogi klawisz będzie uspokajał się w ciągu pięciu milisekund, więc jeśli oprogramowanie skanujące klawiaturę przygląda się temu klawiszowi co dziesięć milisekund, więc kontroler będzie faktycznie gubił odbijanie klawisza.

Zanotujmy, ze naciśnięty klawisz nie jest wystarczającym powodem generowaniem kodu klawisza. Użytkownik może przetrzymać naciśnięty klawisz wiele dziesiątków milisekund nim go zwolni. Sterownik klawiatury nie musi generować owej sekwencji klawisza za każdym razem kiedy skanuje klawiaturę i znajduje wciśnięty klawisz. Zamiast tego, powinien wygenerować pojedynczą wartość kodu klawisza kiedy klawisz zmienia pozycję z górnej na dolną (operacja naciśnięcia) Po wykryciu, wciśniętego klawisza, mikrokontroler wysyła kod klawisza do PC. Kod klawisza nie odnosi się do kodu ASCII dla tego klawisza, jest to wartość jaką wybrał IBM kiedy po raz pierwszy zaprojektował klawiaturę dla PC.

Klawiatura PC w rzeczywistości generuje dwa kody klawisza dla każdego naciśniętego klawisza. Generuje kod dolny, kiedy klawisz jest naciśnięty i kod górny, kiedy zwalniamy klawisz. Chip mikrokontrolera 8042 przekazuje te kody klawisza do PC, gdzie są przetwarzane przez podprogram obsługi przerwań klawiaturowych. Posiadanie oddzielnego kodu dolnego i górnego jest ważne ponieważ pewne klawisze (takie jak shift, control i alt) mają znaczenie tylko jeśli są wciśnięte. Poprzez generowanie górnych kodów dla wszystkich klawiszy, klawiatura zapewnia ,że podprogram obsługi przerwań klawiaturowych wie jakie klawisze naciśnięto podczas gdy użytkownik trzyma wciśnięty jeden z tych klawiszy. Poniższa tablica pokazuje kody klawiszy, jakie mikrokontroler przekazuje do PC:

Klawisz	Dół	Góra		Klawisz	Dół	Góra		Klawisz	Dół	Góra		Klawisz	Dół	Góra
Esc	1	81		[{	1A	9A		, <	33	B3		**center**	4C	CC
1 !	2	82		] }	1B	9B		. >	34	B4		**right**	4D	CD
2 @	3	83		Enter	1C	9C		/ ?	35	B5		+	4E	CE
3 #	4	84		Ctrl	1D	9D		R shift	36	B6		**end**	4F	CF
4 $	5	85		A	1E	9E		*PrtScr	37	B7		**down**	50	D0
5 %	6	86		S	1F	9F		alt	38	B8		**pgdn**	51	D1
6 ^	7	87		D	20	A0		Space	39	B9		**ins**	52	D2
7 &	8	88		F	21	A1		CAPS	3A	BA		**del**	53	D3
8 *	9	89		G	22	A2		F1	3B	BB		/	E0 35	B5
9 (	0A	8A		H	23	A3		F2	3C	BC		**enter**	E0 1C	9C
0)	0B	8B		J	24	A4		F3	3D	BD		F11	57	D7
- _	0C	8C		K	25	A5		F4	3E	BE		F12	58	D8
= +	0D	8D		L	26	A6		F5	3F	BF		ins	E0 52	D2
Bksp	0E	8E		; :	27	A7		F6	40	C0		del	E0 53	D3
Tab	0F	8F		' "	28	A8		F7	41	C1		home	E0 47	C7
Q	10	90		` ~	29	A9		F8	42	C2		end	E0 4F	CF
W	11	91		L shift	2A	AA		F9	43	C3		pgup	E0 49	C9
E	12	92		\ |	2B	AB		F10	44	C4		pgdn	E0 51	D1
R	13	93		Z	2C	AC		NUM	45	C5		left	E0 4B	CB
T	14	94		X	2D	AD		SCRL	46	C6		right	E0 4D	CD
Y	15	95		C	2E	AE		**home**	47	C7		up	E0 48	C8
U	16	96		V	2F	AF		**up**	48	C8		down	E0 50	D0
I	17	97		B	30	B0		**pgup**	49	C9		R alt	E0 38	B8
O	18	98		N	31	B1		-	4A	CA		R ctrl	E0 1D	9D
P	19	99		M	32	B2		**Lefy**	4B	CB		Pause	E1 D1 45 E1 9D C5	-

Tablica 72: Kody klawiszy klawiatury PC (w hex)

Klawisze pogrubioną czcionką są to klawisze z klawiatury numerycznej. Zauważmy ,że pewne klawisze przekazują dwa lub więcej kodów klawiszy do systemu. Klucze które przekazują więcej niż jeden kod klawisza to nowe klawisze, dodane kiedy IBM zaprojektował 101 klawiszową rozszerzoną klawiaturę.

Kiedy kod klawisza przychodzi do PC, drugi mikrokontroler odbiera ten k0d, konwertuje na kod klawisza, robi go dostępnym na porcie I/O 60h a potem wysyła przerwanie do procesora a potem zostawia go w ISR'ze klawiatury pobierającym kod klawisza z portu I/O.

Podprogram obsługi przerwań klawiaturowych (int 9) odczytuje kod klawisza z portu wejściowego klawiatury i odpowiednio przetwarza ten kod klawisza. Zauważmy, że kod klawisza system odbiera z mikrokontrolera klawiatury jako pojedynczą wartość, pomimo, że pewne klawisze reprezentują do czterech różnych wartości. Na przykład klawisz „A" na klawiaturze może stworzyć A, a, ctrl-A lub alt-A. Rzeczywisty kod w systemie zależy od bieżącego stanu klawiszy modyfikujących (shift, ctrl, alt, capslock i numlock). NA przykład, jeśli kod klawisza A nadchodzi jako (1Eh) i jest wciśnięty klawisz shift, system tworzy kod ASCII dla dużej litery. Jeśli użytkownik naciśnie wiele klawiszy modyfikujących , system zadziała według priorytetów od najniższego do najwyższego jak następuje:

- Żaden klawisz modyfikujący nie wciśnięty
- Numlock / Capslock (takie samo pierwszeństwo, najniższy priorytet)
- Shift
- Ctrl

- Alt (najwyższy priorytet)

Numlock i Capslock wpływają na różne zbiory klawiszy, więc nie ma żadnych niejasności wynikających z ich równego pierwszeństwa w powyższym zestawieniu. Jeśli użytkownik naciśnie dwa klawisze modyfikujące w tym samym czasie, system rozpozna tylko klawisz modyfikujący o najwyższym priorytecie. Na przykład, jeśli użytkownik naciśnie klawisze cytr i alt, system rozpozna tylko klawisz alt. Klawisze numlock, capslock i shift są specjalnymi przypadkami. Jeśli numlock lub capslock są aktywne, naciśnięcie shift uczyni je nieaktywnymi. Podobnie jeśli numlock lub capslock są nieaktywne, naciśnięcie klawisza shift skutecznie „aktywuje" te modyfikatory.

Nie wszystkie modyfikatory są poprawne dla każdego klawisza. Na przykład ctrl – 8 jest niepoprawną kombinacją. Podprogram obsługi przerwania klawiaturowego zignoruje wszystkie kombinacje naciśnięć klawiszy z niepoprawnymi klawiszami modyfikującymi. Z nieznanych powodów IBM zadecydował o uczynieniu pewnych kombinacji poprawnymi a innych niepoprawnymi. Na przykład ctrl – left i ctrl - right są poprawne ale ctrl – up i ctrl –down już nie. Jak poprawić ten problem zobaczymy trochę później.

Klawisze shift, alt i ctrl są aktywnymi modyfikatorami. To znaczy, modyfikacja naciśniętego klawisza wystąpi tylko, kiedy użytkownik trzyma wciśnięty jeden z tych klawiszy modyfikujących. ISR klawiatury śledzi czy klawisze te są wciśnięte czy nie. Przeciwnie, klawisze numlock, scroll lock i capslock są modyfikatorami przełączającymi. ISR klawiaturowy odwraca powiązane bity za każdym razem kiedy widzi dolny kod następujący po kodzie górnym, dla tych klawiszy.

Większość klawiszy klawiatury PC odpowiada znakom ASCII. Kiedy ISR klawiaturowy napotka taki znak, tłumaczy go na 16 bitową wartość, której najmniej znaczący bajt jest kodem ASCII a bardziej znaczący bajt kodem klawiaturowym klawisza. Na przykład, naciskając „A", bez modyfikatora, z shift i z control tworzymy odpowiednio 1E61h, 1E41h i 1E01h, („a", „A" i ctrl-A) Wiele sekwencji klawiszy nie ma odpowiedników kodów ASCII. Na przykład klawisze funkcyjne, klawisz sterujący kursorem i sekwencja klawisza alt nie mają odpowiedników kodów ASCII. Dla tych specjalnych, rozszerzonych kodów, ISR klawiaturowy przechowuje zero w mnij znaczącym bajcie (gdzie zazwyczaj jest kod ASCII) a rozszerzony kod idzie do bardziej znaczącego bajtu. Kod rozszerzony jest zazwyczaj, chociaż z pewnością nie zawsze, kodem klawiaturowym dla tego klawisza.

Jedyny problem z kodem rozszerzonym jest taki, że wartość zero jest poprawnym znakiem ASCII (znak NUL). Dlatego też nie możemy bezpośrednio wprowadzić znaku NUL do aplikacji. Jeśli aplikacja musi wprowadzić znak NUL, IBM ma zarezerwowane miejsce w pamięci dla rozszerzonego kodu 0300h (ctrl-3) . Aplikacja wyraźnie musi skonwertować ten rozszerzony kod do znaku NUL (w rzeczywistości tylko rozpoznać bardziej znaczący bajt wartości 03 ponieważ najmniej znaczący bajt jest już znakiem NUL). Na szczęście, bardzo niewiele programów musi zezwalać na wprowadzanie znaku NUL z klawiatury, więc ten problem jest rzadkością.

Następująca tablica pokazuje kod klawiaturowy i rozszerzony ISR klawiaturowego generowane dla aplikacji w odpowiedzi na naciśnięcie klawisza z różnymi modyfikatorami Kody rozszerzone są pogrubione. Wszystkie wartości (z wyjątkiem kolumny kodów klawiaturowych) przedstawiają osiem najmniej znaczących bitów 16 bitowego kodu. Bardziej znaczący bajt pochodzi z kolumny kodów klawiaturowych.

Klawisz	Kod klawiaturowy	ASCII	Shift	Ctrl	Alt	Num	Caps	Shift Caps	Shift Num
Esc	01	1B	1B	1B		1B	1B	1B	1B
1 !	02	31	21		**7800**	31	31	31	31
2 @	03	32	40	**0300**	**7900**	32	32	32	32
3 #	04	33	23		**7A00**	33	33	33	33
4 $	05	34	24		**7B00**	34	34	34	34
5 %	06	35	25		**7C00**	35	35	35	35
6 ^	07	36	5E	1E	**7D00**	36	36	36	36
7 &	08	37	26		**7E00**	37	37	37	37
8 *	09	38	2A		**7F00**	38	38	38	38
9 (	0A	39	28		**8000**	39	39	39	39
0)	0B	30	29		**8100**	30	30	30	30
- _	0C	2D	5F	1F	**8200**	2D	2D	5F	5F
= +	0D	3D	2B		**8300**	3D	3D	2B	2B
Bksp	0E	08	08	7F		08	08	08	08
Tab	0F	09	**0F00**			09	09	**0F00**	**0F00**
Q	10	71	51	11	**1000**	71	51	71	51
W	11	77	57	17	**1100**	77	57	77	57

E	12	65	45	05	**1200**	65	45	65	45
R	13	72	52	12	**1300**	72	52	72	52
T	14	74	54	14	**1400**	74	54	74	54
Y	15	79	59	19	**1500**	79	59	79	59
U	16	75	55	15	**1600**	75	55	75	55
I	17	69	49	09	**1700**	69	49	69	49
O	18	6F	4F	0F	**1800**	6F	4F	6F	4F
P	19	70	50	10	**1900**	70	50	70	50
[{	1A	5B	7B	1B		5B	5B	7B	7B
]}	1B	5D	7D	1D		5D	5D	7D	7D
enter	1C	0D	0D	0A		0D	0D	0A	0A
ctrl	1D								
A	1E	61	41	01	**1E00**	61	41	61	41
S	1F	73	53	13	**1F00**	73	53	73	52
D	20	64	44	04	**2000**	64	44	64	44
F	21	66	46	06	**2100**	66	46	66	46
G	22	67	47	07	**2200**	67	47	67	47
H	23	68	48	08	**2300**	68	48	68	48
J	24	6A	4A	0A	**2400**	6A	4A	6A	4A
K	25	6B	4B	0B	**2500**	6B	4B	6B	4B
L	26	6C	4C	0C	**2600**	6C	4C	6C	4C
; :	27	3B	3A			3B	3B	3A	3A
‘ “	28	27	22			27	27	22	22
` ~	29	60	7E			60	60	7E	7E
Lshift	2A								
\ |	2B	5C	7C	1C		5C	5C	7C	7C
Z	2C	7A	5A	1A	**2C00**	7A	5A	7A	5A
X	2D	78	58	18	**2D00**	78	58	78	58
C	2E	63	43	03	**2E00**	63	43	63	43
V	2F	76	56	16	**2F00**	76	56	76	56
B	30	62	42	02	**3000**	62	42	62	42
N	31	6E	4E	0E	**3100**	6E	4E	6E	4E
M	32	6D	4D	0D	**3200**	6D	4D	6D	4D
, <	33	2C	3C			2C	2C	3C	3C
. >	34	2E	3E			2E	2E	3E	3E
/ ?	35	2F	3F			2F	2F	3F	3F
Rshift	36								
* PrtSc	37	2A	INT 5	10		2A	2A	INT 5	INT 5
alt	38								
space	39	20	20	20		20	20	20	20
caps	3a								
F1	3B	**3B00**	**5400**	**5E00**	**6800**	**3B00**	**3B00**	**5400**	**5400**
F2	3C	**3C00**	**5500**	**5F00**	**6900**	**3C00**	**3C00**	**5500**	**5500**
F3	3D	**3D00**	**5600**	**6000**	**6A00**	**3D00**	**3D00**	**5600**	**5600**
F4	3E	**3E00**	**5700**	**6100**	**6B00**	**3E00**	**3E00**	**5700**	**5700**
F5	3F	**3F00**	**5800**	**6200**	**6C00**	**3F00**	**3F00**	**5800**	**5800**
F6	40	**4000**	**5900**	**6300**	**6D00**	**4000**	**4000**	**5900**	**5900**
F7	41	**4100**	**5A00**	**6400**	**6E00**	**4100**	**4100**	**5A00**	**5A00**
F8	42	**4200**	**5B00**	**6500**	**6F00**	**4200**	**4200**	**5B00**	**5B00**
F9	43	**4300**	**5C00**	**6600**	**7000**	**4300**	**4300**	**5C00**	**5C00**
F10	44	**4400**	**5D00**	**6700**	**7100**	**4400**	**4400**	**5DD0**	**5D00**
num	45								
scrl	46								
home	47	**4700**	37	**7700**		37	4700	37	4700
up	48	**4800**	38			38	4800	38	4800
pgup	49	**4900**	39	**8400**		39	4900	39	4900
-	4A	2D	2D			2D	2D	2D	2D
left	4B	**4B00**	34	**7300**		34	4B00	34	4B00

center	4C	**4C00**	35			35	4C00	35	4C00
right	4D	**4D00**	36	**7400**		36	4D00	36	4D00
+	4E	2B	2B			2B	2B	2B	2B
end	4F	**4F00**	31	**7500**		31	4F00	31	4F00
down	50	**5000**	32			32	5000	32	5000
pgdn	51	**5100**	33	**7600**		33	5100	33	5100
ins	52	**5200**	30			30	5200	30	5200
del	53	**5300**	2E			2E	5300	2E	5300

Tablica 73: Kody klawiatury (w hex)

Klawiatura 101 klawiszowa ogólnie rzecz biorąc dostarcza klawisza enter i „/" w bloku klawiatury numerycznej. Chyba, że piszemy własny ISR klawiaturowy int 9, wtedy nie będziemy mogli rozróżniać tych klawiszy od klawiszy z klawiatury głównej. Oddzielny blok sterowania kursorem również generuje taki sam kod rozszerzony co blok numeryczny, z wyjątkiem tego ,że nigdy nie generuje numerycznego kodu ASCII. W przeciwnym razie, nie możemy rozróżnić klawiszy z odpowiadającymi klawiszami na klawiaturze numerycznej (zakładając oczywiście, że numlock jest wyłączony).

ISR klawiaturowy dostarcza specjalnej umiejętności, która pozwala nam wprowadzać kod ASCII dla wciskanych klawiszy bezpośrednio z klawiatury. Robiąc to wciskamy klawisz alt i wpisujemy dziesiętny kod ASCII (0..255) dla znaku z klawiatury numerycznej. ISR klawiaturowy skonwertuje to naciśnięcie klawisza na ośmiobitową wartość, dokładając zero w bardziej znaczącym bajcie do znaku i używa tego jako kodu znaku.

ISR klawiaturowy wprowadza 16 bitową wartość do bufora roboczego PC Systemowy bufor roboczy jest cykliczną kolejką która używa następujących zmiennych:

```
40:1A   - HeadPtr       word    ?
40:1C   -  TailPtr      word    ?
40:1E   -   Buffer      word    16  dup (?)
```

ISR klawiaturowy wprowadza dane pod lokację wskazywaną przez TailPtr. Funkcja klawiaturowa BIOS usuwa znaki z lokacji wskazywanej przez zmienną HeadPtr. Te dwa wskaźniki prawie zawsze zawierają offset tablicy Buffer. Jeśli te dwa wskaźniki są równe, bufor roboczy jest pusty. Jeśli wartość w HeadPtr jest dwa razy większa niż wartość w TailPtr (lub HeadPtr zawiera 1Eh a TailPtr 3Ch) wtedy bufor jest pełny a ISR klawiaturowy będzie odrzucał dodatkowe naciskania klawiszy.

Zauważmy, że zmienna TailPtr zawsze wskazuje następną dostępną lokację w buforze roboczym. Ponieważ nie ma zmiennej„licznik" dostarczającej liczby wejść w buforze, musimy zawsze zostawić jedno wolne wejście w obszarze bufora; to oznacza, że bufor roboczy może przechowywać tylko 15 a nie 16 naciśnięć klawisza.

Oprócz bufora roboczego, BIOS utrzymuje kilka innych zmiennych powiązanych z klawiaturą w segmencie 40h. Poniższa tablica pokazuje te zmienne i ich zawartość:

Nazwa	Adres	Rozmiar	Opis
KbdFlags1 (flagi modyfikatorów)	40:17	Bajt	Bajt ten utrzymuje bieżący stan klawiszy modyfikujących na klawiaturze. Bity mają następujące znaczenie: bit 7: wprowadzono tryb przełączania bit 6: przełączony Capslock (1 = włączony) bit 5: przełączenie Numlock (1 = włączony) bit 4: przełączony Scroll lock (1 = włączony) bit 3: klawisz Alt (1 = wciśnięty) bit 2: klawisz Ctrl (1 = wciśnięty) bit 1: Lewy shift (1 = włączony) bit 0: prawy shift (1 = włączony)
KbdFlags2	40:18	Bajt	Określa czy przełączany klawisz jest obecnie w dole bit 7: Klawisz Insert (w dole jeśli 1 bit 6: Klawisz Capslock (jak wyżej) bit 5:Klawisz Numlock (jak wyżej) bit 4: Klawisz Scroll lock (jak wyżej) bit 3: Stan pauzy (ctrl+Numlock) jeśli 1 bit 2: Klawisz SysReq (jak wyżej) bit 1: Klawisz lewy alt (jak wyżej) bit 0: Klawisz lewy ctrl (jak wyżej)

AltKpd	40:19	Bajt	BIOS używa go do obliczania kodu ASCII dla sekwencji alt – blok klawiszy
BufStart	40:80	Słowo	Offset startowy bufora klawiatury (1Eh). Notka: zmienna ta nie jest wspierana przez wiele systemów, bądźmy ostrożni stosując ją
BufEnd	40:82	Słowo	Offset końcowy bufora klawiatury (3Eh).
KbdFlags3	40:96	Bajt	Różne flagi klawiatury bit 7: odczyt ID klawiatury w toku bit 6: ostatni znak jest pierwszym znakiem ID klawiatury bit 5: wymuszenie numlock przy resecie bit 4: 1 jeśli klawiatura 101 klawiszowa, 0 jeśli 83/84 bit 3: naciśnięty prawy klawisz alt jeśli 1 bit 2: naciśnięty prawy klawisz ctrl jeśli 1 bit 1: ostatnim kodem klawiaturowym był E0h bit 0: Ostatnim kodem klawiaturowym był E1h
KbdFlags4	40:97	Bajt	Więcej różnych flag klawiatury bit 7: błąd przesyłania klawiatury bit 6: aktualizacja wskaźnika trybu bit 5: flaga odbiorcza ponownego wysyłania bit 4: potwierdzenia odbioru bit 3: musi zawsze być zerem bit 2: LED Capslocka (1 = włączona) bit 1: LED Numlocka (1 = włączona) bit 0: LED Scroll lock (1 = włączona)

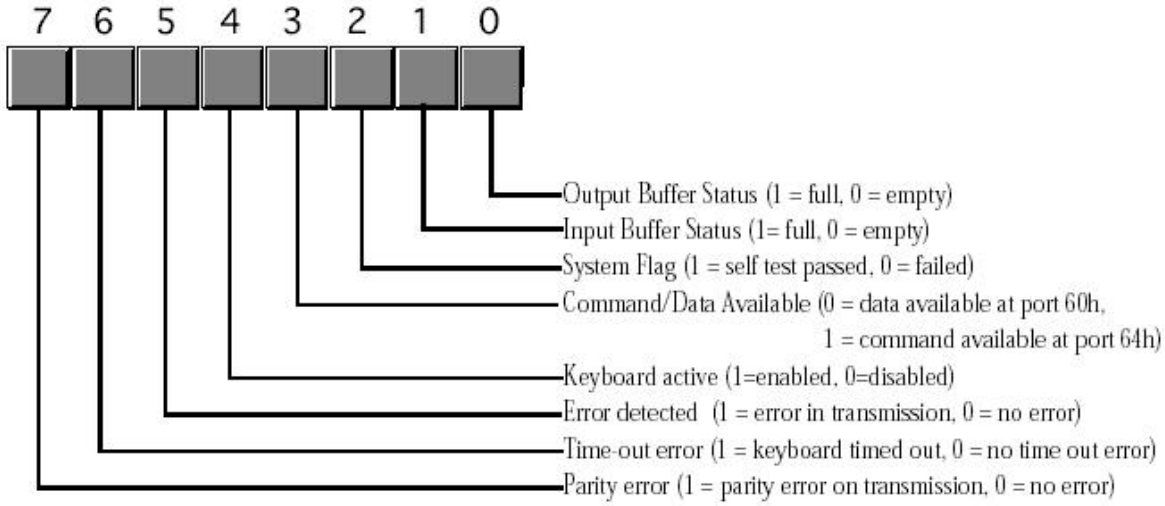
Tablica 74: Zmienne BIOS'a powiązane z klawiaturą

Krótki komentarz do KbdFlags1 i KbdFlags3. Bity od zera do dwóch zmiennej KbdFlags3 są bieżącym ustawieniem BIOS'a dla diod LED na klawiaturze, okresowo BIOS porównuje te wartości dal capslocka, numlocka i scroll locka w KbdFlags1 z tymi trzema bitami w KbdFlags4. Jeśli nie są zgodne, BIOS będzie wysyłał właściwe polecenie do klawiatury aktualizując LED'y i zmieniając wartości w zmiennej KbdFlags4 aby system był spójny. Dlatego też maskując nowe wartości dla numlock, capslock lub scroll lock, BIOS będzie automatycznie modyfikował KbdFlags4 i ustawiał odpowiednio LED'y .

20.2 INTERFEJS SPRZĘTOWY KLAWIATURY

IBM użył bardzo prostego sprzętowego projektu dla portu klawiatury w oryginalnych maszynach PC i PC/XT. Kiedy wprowadzili PC/AT, IBM kompletnie przeprojektował interfejs między PC a klawiaturą. Od tego czasu prawie każdy model PC i klony PC mają ten standardowy interfejs klawiatury. Chociaż IBM rozszerzył zdolności sterownika klawiatury, kiedy wprowadził swój system PS/2, modele PS/2 są jeszcze kompatybilne w górę z PC/AT. Ponieważ jest jeszcze kilka oryginalnych PC w użyciu dzisiaj (a kilku ludzi pisze oryginalne programy dla nich), zignorujemy oryginalny interfejs klawiatury PC i skoncentrujemy się na AT i późniejszych projektach.

Są dwa mikrokontrolery klawiatury, które komunikują się z systemem – jeden na płycie głównej PC (mikrokontroler zintegrowany) i jeden wewnątrz obudowy klawiatury (mikrokontroler klawiatury). Komunikacja z zintegrowanym mikrokontrolerem następuje przez port I/O 64h.Odczyt tego bajtu dostarczy stanu kontrolera klawiatury. Zapisując do tego bajtu wysyłamy polecenie do zintegrowanego mikrokontrolera. Organizacja bajtu stanu



On-Board 8042 Keyboard Microcontroller Status Byte (Read Port 64h)

Komunikacja z mikrokontrolerem w klawiaturze następuje przez adresy I/O 60h i 64h. Bity zero i jeden w bajcie stanu portu 64h dostarczają sterowania z potwierdzeniem dla tych portów. Przed zapisaniem danych do tych portów, bit zero portu 64h musi być wyzerowany; dana jest dostępna do odczytu z portu 60h, kiedy bit jeden portu 64 zawiera jeden. Klawiatura włącza i wyłącza bity bajtu poleceń (port 64h) określa czy klawiatura jest aktywna i czy klawiatura będzie przerywać system kiedy użytkownik naciska (lub zwalnia) klawisz, itd.

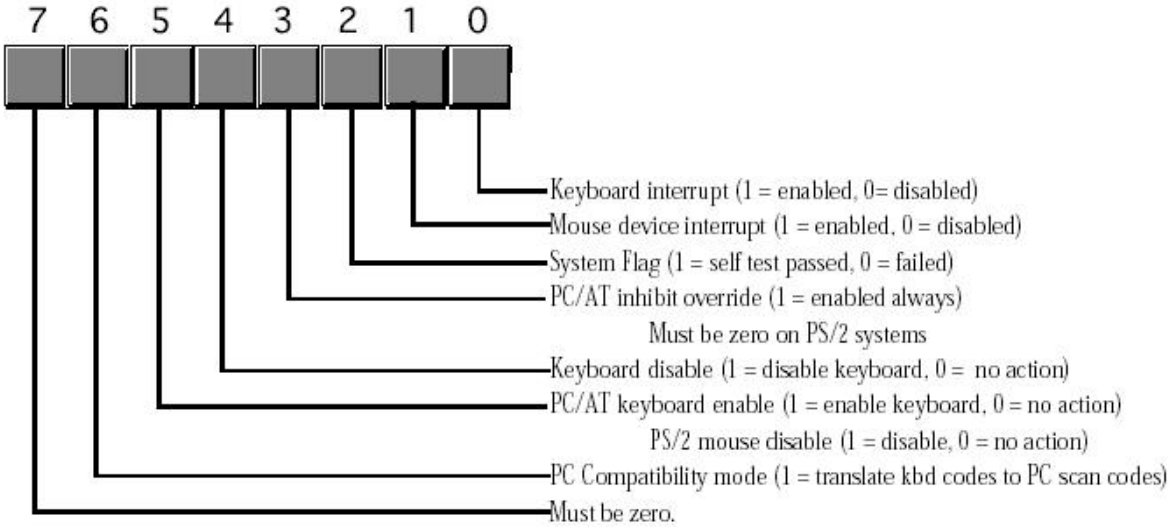
Bajty zapisane do portu 60h są wysyłane do mikrokontrolera klawiatury a bajty zapisane do portu 64h są wysyłane do mikrokontrolera zintegrowanego. Bajty odczytane z portu 60h ogólnie rzecz biorąc pochodzą z klawiatury, chociaż możemy oprogramować zintegrowany mikrokontroler aby również zwracał pewne wartości spod tego portu. Poniższa tabele pokazują polecenia wysyłane do mikrokontrolera klawiatury i wartości jakich się możemy się spodziewać. Pokazują również dostępne polecenia jakie możemy zapisać do portu 64h:

Wartość (w hex)	Opis
20	Przekazuje bajt poleceń kontrolera klawiatury do systemu jako kod klawiaturowy do portu 60h
60	Kolejny bajt zapisuje do portu 60h będzie przechowywany w bajcie poleceń kontrolera klawiatury
A4	Testuje czy jest zainstalowane sprawdzanie praw dostępu (tylko PS/2) . Wynik ponownie wraca do portu 60h. 0FAh oznacza ,że jest zainstalowany, 0F1h ,że nie
A5	Przekazanie hasła (tylko PS/2). Start odbierania hasła. Kolejna sekwencja kodów klawiaturowych zapisana do portu 60h, zakończona bajtem zerowym, jest nowym hasłem.
A6	Dopasowanie hasła. .Znaki z klawiatury są porównywane z hasłem dopóki nie wystąpi dopasowanie.
A7	Blokuje myszkę (tylko PS/2). Ustawiamy piąty bit bajtu poleceń.
A8	Włącza myszkę (tylko PS/2) Zerujemy piąty bit bajtu poleceń.
A9	Testuje myszkę. Zwraca 0 jeśli jest OK, 1 lub 2 jeśli jest zablokowany zegar, 3 lub 4 jeśli zablokowana jest linia danych. Wynik zwracany w porcie 60h
AA	Inicjacja programu samostartującego . Zwraca 55h w porcie 60h jeśli powodzenie
AB	Test interfejsu klawiatury. Testuje interfejs klawaitury. Zwraca 0 jeśli OK, 1 lub 2 jeśli jest zablokowany zegar, 3 lub 4 jeśli zablokowana linia danych. Wynik wraca do portu 60h
AC	Diagnostyka. Zwraca 16 bajtów z chipu mikrokontrolera klawiatury. Nie dostępny w systemach PS/2.
AD	Zablokowanie klawiatury. Operacje ustawiają bit cztery rejestru poleceń
AE	Odblokowanie klawiatury. Operacje zerują bit cztery rejestru poleceń

C0	Odczyt wejściowego portu klawiatury z portu 60h. Ten port wejściowy zawiera poniższe wartości: bit 7: Hamowanie przełączania klawiszy (0= zahamowane, 1 = odblokowane) bit 6: wyświetlanie (0= kolor, 1= mono) bit 5: wytwarzanie zworki bit 4: płyta systemowa RAM (zawsze 10 bit 0-3; niezdefiniowane
C1	Kopiowanie bitów 0-3 portu wejściowego do bitów stanu 4-7 (tylko PS/2)
C2	Kopiowanie bitów 4-7 portu wejściowego do bitów stanu 4-7 portu (tylko PS/2)
D0	Kopiowanie wartości portu wyjściowego mikrokontrolera do portu 60h (zobacz poniższą definicję)
D1	Zapis następnego bajtu danych zapisanego do portu 60h portu wyjściowego mikrokontrolera: bit 7: Dana klawiatury bit 6: zegar klawiatury bit 5: flaga pustego bufora wejściowego bit 4: flaga pełnego bufora wyjściowego bit 3: niezdefiniowany bit 2: niezdefiniowany bit 1: linia Gate A20 bit 0: reset systemu (jeśli zer0) Notka: zapisanie zera do bity zero zresetuje maszynę Zapis jeden do bitu jeden połączy adres lini 19 i 20 na szynie adresowej PC
D2	Zapis bufora klawiatury. Kontroler klawiatury zwraca kolejną wartość wysłaną do portu 60h jak gdyby naciśnięcie klawisza tworzyło tą wartość (tylko PS/2)
D3	Zapis bufora myszy. Kontroler klawiatury zwraca kolejną wartość wysyłaną do portu 60h jak gdyby działanie myszy tworzyło tą wartość
D4	Zapis kolejnego bajtu danych (60h) do myszki (pomocniczo) (tylko PS/2)
E0	Odczyt testu wejściowego. Zwraca w porcie 60h stan lini szeregowej klawiatury. Bit zero zawiera zegar wejściowy klawiatury, bit jeden zawiera daną wejściową klawiatury
Fx	Impuls portu wyjściowego (zobacz definicję D1). Bity 0-3 bajtu poleceń kontrolera klawiatury są impulsowane do portu wyjściowego. Reset systemu jeśli bit zero jest zerem.

Tablica 75: Polecenia zintegrowanego kontrolera klawiatury (Port 64h)

Polecenia 20h i 60h pozwalają odczytać i zapisać bajt poleceń kontrolera klawiatury. Bajt ten jest wewnątrz zintegrowanego mikrokontrolera i ma następujące rozmieszczenie:



System przekazuje bajty zapisane do portu I/O 60h bezpośrednio do mikrokontrolera klawiatury. Bit zero rejestru stanu musi zawierać zero przed zapisaniem danej do tego portu. Polecenia klawiatury są rozpoznawane:

Wartość (w hex)	Opis
ED	Przesyłanie bitów LED. Kolejny bajt zapisany do portu 60h aktualizuje LED'y na klawiaturze. Bity zawierają: bity 3-7: Muszą być zerowe bit 2: LED Capslock (1 = włączona, 0 = wyłączona) bit 1: LED Numlock (1 = włączony, 0 = wyłączony) bit 0: LED Scroll lock (1= włączona, 0 = wyłączona)
EE	Polecenia echa. Zwraca 0Eeh w porcie 60h jako pomoc diagnostyczną
F0	Wybiera alternatywny zbiór kodów klawiaturowych (tylko PS/2). Kolejny bajt zapisany do portu 60h wybiera jedną z opcji: 00: Raportuje aktualny zbiór kodów klawiaturowych (kolejny bajt odczytany z portu 60h) 01: Wybór zbioru kodów klawiaturowych #1 (standardowy zbiór kodów PC/AT 02: Wybór zbioru kodów klawiaturowych # 2 03: Wybór zbioru kodów klawiaturowych # 3
F2	Wysyłanie dwu bajtowego kodu ID klawiatury jako kolejnych dwóch bajtów odczytanych z portu 60h (tylko PS/2)
F3	Ustawienie opóźnienia autopowtarzania i częstotliwości powtarzania. Kolejny bajt zapisany do portu 60h określa częstotliwość: bit 7 : musi być zerem bity 5-6: Opóźnienie 00-1/ 4 sek, 01 –1/ 2 sek, 10 3 / 4 sek, 11 – 1 sek bity 0-4: Częstotliwość powtarzania 0 –ok. 30 znaków / sek d0 1Fh – ok. 2 znaki / sek
F4	Włączenie klawiatury
F5	Reset włączenia i oczekiwanie na polecenie włączenia
F6	Reset włączenia i początek skanowania klawiatury
F7	Uczynienie wszystkich klawiszy automatycznie powtarzanymi (tylko PS/2)
F8	Ustawienie wszystkich klawiszy do generowania kodu górnego i dolnego (tylko PS/2)
F9	Ustawienie wszystkich klawiszy do generowania tylko kodu górnego (tylko PS/2)
FA	Ustawienie wszystkich klawiszy na autopowtarzanie I generowanie tylko kodu górnego (tylko PS/2=
FB	Ustawienie pojedynczych klawiszy na autopowtarzanie. Kolejny bajt zawiera kod klawiaturowy żądanego klawisza (tylko PS/2)
FC	Ustawienie pojedynczego klawisza do generowanie kodów górnego i dolnego. Kolejny bajt zawiera kod klawiaturowy żądanego klawisza
FD	Ustawienie pojedynczego klawisza do generowania tylko kodów dolnych. Kolejny bajt zawiera kod klawiaturowy żądanego kodu
FE	Ponowne wysłanie ostatniego wyniku. Używamy tego polecenia jeśli

	wystąpił błąd odbioru danej
FF	Reset klawiatury w stanie włączenia i start programu samo startującego

Tablica 76: Polecenia mikrokontrolera klawiatury (port 60h)

Poniższy, krótki program demonstruje jak wysłać polecenia do kontrolera klawiatury. Ten mały TSR pokazuje „pokaz świateł” LED na klawiaturze

```
; LEDSHOW.ASM
;
; Ten krótki TSR tworzy pokaz świateł z LED na klawiaturze. Ten kod nie implementuje złożonego programu
; jaki możemy usuwać ten TSR raz zainstalowany. Zobaczmy rozdział o programach rezydentnych jak
; można to zrobić.
;
; cseg i EndResident muszą wystąpić przed segmentem biblioteki standardowej!

cseg            segment para  public  'code'
cseg            ends

; Oznaczamy segment, znajdujemy koniec sekcji rezydentnej

EndResident     segment para  public 'Resident'
EndResident     ends

                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list

byp             equ     < byte ptr >

cseg            segment para  public  'code'
                assume  cs:cseg, ds:cseg

; SetCmd-       Wysyła bajt poleceń w rejestrze AL do chipu mikrokontrolera klawiatury 8042
;               (rejestr poleceń portu 64h)

SetCmd          proc    near
                push    cx
                push    ax                      ;zachowujemy wartość polecenia
                cli                             ; region krytyczny, żadnych przerwań

; Czekamy dopóki 8042 nie przetworzy bieżącego polecenia

                xor     cx, cx                  ; zezwolenie na 65,536 razy przejść pętle
Wait4Empty:     in      al., 64                 ;odczyt rejestru stanu klawiatury
                test    al., 10b                ; bufor wejściowy pełny?
                loopnz  Wait4Empty              ; jeśli talk, czekamy na opróżnienie

; Okay, wysyłamy polecenie do 8042:

                pop     ax                      ; wyszukanie polecenia
                out     64h, al.
                sti                             ; okay, przerwania mogą być ponownie
                pop     cx
                ret
SetCmd          endp

; SendCmd-      Podprogram wysyła polecenie lub bajt danych do portu danych klawiatury (port 60h)
```

```
SendCmd     proc    near
            push    ds.
            push    bx
            push    cx
            mov     cx, 40h
            mov     ds., cx
            mov     bx, ax                  ;zachowanie bajtu danych

            mov     al., 0Adh               ;zablokowanie klawiatury
            call    SetCmd
            cli                             ;blokowanie przerwań

; Czekamy dopóki 8042 przetworzy bieżące polecenie

            xor     cx, cx                  ; zezwolenie na 65,536 razy przejść pętle
Wait4Empty: in      al., 64                 ;odczyt rejestru stanu klawiatury
            test    al., 10b                ; bufor wejściowy pełny?
            loopnz  Wait4Empty              ; jeśli talk, czekamy na opróżnienie

; Okay, wysyłamy daną do portu 60h

            mov     al., bl
            out     60h, al.
            mov     al, 0Aeh                ;ponownie włączamy klawiaturę
            call    SetCmd
            sti                             ;zezwolenie na przerwania

            pop     cx
            pop     bx
            pop     ds.
            ret
SendCmd     endp

;SetLEDs-   Zapisuje wartość w AL. do LED'ów na klawiaturze. Bity 0..2 odpowiadają odpowiednio
;           scroll, num i caps lock

SetLEDs     proc    near
            push    ax
            push    cx
            mov     ah, al                  ;zachowanie bitów LED
            mov     al., 0Edh               ; 8042 ustawia polecenia LED
            call    SendCmd                 ;wysłanie polecenia do 8042
            mov     al., ah                 ;pobranie bajtu parametru
            call    SendCmd                 ;wysłanie parametru do 8042

            pop     cx
            pop     ax
            ret
SetLEDs     endp

;MyInt1C-   Co 1/ 4 sekundy (co czwarte wywołanie) podprogram ten obraca LED'y tworząc
;           interesujący pokaz świateł

CallsPerInt equ     4
CallCnt     byte    CallsPerInt
LEDIndex    word    LEDTable
LEDTable    byte    111b, 110b, 101b, 011b, 111b ,110b, 101b, 011b
            byte    111b, 110b, 101b, 011b, 111b, 110b, 101b, 011b
            byte    111b, 110b, 101b, 011b, 111b, 110b, 101b, 011b
            byte    111b, 110b, 101b, 011b, 111b, 110b, 101b, 011b
```

```
                byte    000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b
                byte    000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b
                byte    000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b
                byte    000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b

                byte    000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b
                byte    000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b
                byte    000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b
                byte    000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b

                byte    010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b
                byte    010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b
                byte    010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b
                byte    010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b

                byte    000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b
                byte    000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b
                byte    000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b
                byte    000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b

TableEnd        equ     this byte

OldInt1C        dword   ?

MyInt1C         proc    far
                assume  ds:cseg

                push    ds
                push    ax
                push    bx

                mov     ax, cs
                mov     ds, ax

                dec     CallCnt
                jne     NotYet
                mov     CallCnt, CallsPerInt
                mov     bx, LEDIndex
                mov     al, [bx]
                call    SetLEDs
                inc     bx
                cmp     bx, offset  TableEnd
                jne     SetTbl
                lea     bx, LEDTable
SetTbl:         mov     LEDIndex, bx
NotYet:         pop     bx
                pop     ax
                pop     ds
                jmp     cs:OldInt1C
MyInt1C         endp

Main            proc

                mov     ax, cseg
                mov     ds, ax

                print
                byte    "LED Light Show", cr, lf
                byte    "Installing….", cr, lf,0
```

```
; Aktualizujemy wektor przerwania 1C. Zauważmy, że powyższe instrukcje czynią cseg bieżącym
; segmentem  danych, więc możemy przechować  starą wartość INT 1Ch bezpośrednio w zmiennej
; OldInt1C

                cli                                     ;wyłączamy przerwania
                mov     ax, 0
                mov     es, ax
                mov     ax, es:[1Ch*4]
                mov     word  ptr OldInt1C, ax
                mov     ax, es:[1Ch*4+2]
                mov     word  ptr OldInt1C+2, ax
                mov     es:[1Ch*4], offset  MyInt1C
                mov     es:[1Ch*4+2], cs
                sti                                     ; włączamy przerwania

; jedyna rzecz jaka pozostała to zakończenie i pozostanie w pamięci

                print
                byte    „Installed”, cr, lf, 0

                mov     ah, 62h                         ;pobranie  wartości PSP programu
                int     21h

                mov     dx, EndResident                 ;obliczenie rozmiaru programu
                sub     dx, bx
                mov     ax, 3100h                       ;polecenie TSR DOS’a
                int     21h
Main            endp
cseg            ends

sseg            segemnt para stack ‘stack’
stk             db      1024   dup (“stack”)
sseg            ends

zzzzzzseg       segment para  public ‘zzzzzz’
LastBytes       db      16  dup  (?)
Zzzzzzseg       ends
                end     Main
```

Mikrokontroler klawiatury również wysyła dane do zintegrowanego mikrokontrolera do przetworzenia i udostępnienie systemowi przez port 60h. Większość z tych wartości jest kodami klawiaturowymi naciskanych klawiszy (kody górny lub dolny), ale klawiatura przekazuje również kilka innych wartości . Dobrze zaprojektowany podprogram obsługi przerwań klawiaturowych powinien móc obsłużyć (lub przynajmniej zignorować) wartości kodów nie klawiaturowych. W szczególności , program który wysyła polecenia do klawiatury musi móc obsłużyć ponowne wysłanie i potwierdzenie poleceń, które mikrokontroler klawiatury zwraca w porcie 60h. Mikrokontroler klawiatury wysyła do systemu następujące wartości :

Wartość (hex)	Opis
00	Przepełnienie danych. System wysyła bajt zero jako ostatnią wartość kiedy przepełni się wewnętrzny bufor kontrolera klawiatury.
1..58 81..D8	Kody klawiaturowe dla naciśniętych klawiszy. Wartości dodatnie są kodami dolnymi, wartości ujemne (ustawiony najbardziej znaczący bit) są kodami górnymi
83AB	ID klawiatury zwracane w odpowiedzi na polecenie F2 (tylko PS/2)
AA	Zwracane podczas podstawowego testu bezpieczeństwa po resecie .Również górny kod dla lewego klawisza Shift
EE	Zwracane przez polecenie ECHO
F0	Przedrostek pewnych górnych kodów

FA	Potwierdzenie klawiatury dla poleceń klawiaturowych innych niż ponowne wysłanie lub ECHO
FC	Niepowodzenie podstawowego testu bezpieczeństwa (tylko PS/2)
FD	Niepowodzenie diagnostyki (nie dostępne na PS/2)
FE	Ponowne wysłanie. Klawiatura wymaga od systemu ponownego wysłania ostatniego polecenia
FF	Błąd klawisza (tylko PS/2)

Tablica 77: Transmisja z klawiatury do systemu

Zakładając, że mamy nie zablokowane przerwania klawiaturowe (zobacz bajt poleceń kontrolera klawiatury), każda wartość mikrokontrolera klawiatury wysyłana do systemu przez port 60h będzie generowała przerwanie na lini jeden IRQ (int 9). Dlatego też podprogram obsługi przerwań klawiaturowych zazwyczaj obsługuje wszystkie powyższe kody. Jeśli aktualizujemy int 9, nie zapomnijmy wysłać sygnału zakończenia przerwania (EOI) do PIC 8259 na końcu naszego kodu ISR'a. Również nie zapomnijmy, że możemy odblokować lub zablokować przerwanie klawiatury z pod 8259A

Ogólnie rzecz biorąc, nasze aplikacje nie powinny mieć bezpośredniego dostępu do sprzętu klawiatury. Robiąc tak, prawdopodobnie uczynimy nasz software niekompatybilnym z oprogramowaniem użytkowym takim jak rozszerzona klawiatura (makro pogramy klawiatury), oprogramowanie pop-up i inne rezydentne programy, które czytają z klawiatury lub wprowadzają dane do systemowego bufora roboczego. Na szczęście, DOS i BIOS dostarczają doskonałego zbioru funkcji do odczytu i zapisu danych klawiaturowych. Nasze programy będą dużo bardziej stabilne jeśli będziemy przestrzegali stosowania tych funkcji. Dostęp bezpośredni do sprzętu klawiatury powinien być pozostawiony ISR'owi klawiaturowemu i tych poprawek klawiatury i programów pop-up, które koniecznie muszą komunikować się bezpośrednio ze sprzętem.

20.3 INTERFEJS DOS KLAWIATURY

MS-DOS dostarcza kilku funkcji do odczytu z T znaczy, że gubimy informacje kodów klawiaturowych podprogramu obsługi przerwań klawiaturowych zachowanych w buforze roboczym.

Jeśli naciśniemy klawisz, który ma rozszerzony kod zamiast kodu ASCII, MS-DOS zwróci dwa kody klawiszy. Najpierw funkcja DOS zwróci wartość zero. To mówi nam, że musimy ponownie wywołać podprogram pobrania znaku. Kod MS-DOS zwracany w drugim wywołaniu jest rozszerzonym kodem klawisza.

Zauważmy, że podprogramy Biblioteki Standardowej wywołują MS-DOS do odczytu znaków z klawiatury. Dlatego też, podprogram getc Biblioteki Standardowej również zwraca kod klawisza w ten sposób. Podprogramy gets i getsm odrzucają każde nie –ASCII uderzenie klawisza ponieważ nie może być dobrą rzeczą wprowadzenie bajtów zerowych w środek ciągu zakończonego zerem.

20.4 INTERFEJS BIOS KALWIATURY

Chociaż MS-DOS dostarcza stosownego zbioru podprogramów do odczytu kodów ASCII i znaków rozszerzonych z klawiatury, BIOS PC dostarcza dużo lepszych udogodnień wejścia klawiatury Co więcej, jest dużo interesujących powiązanych zmiennych w obszarze BIOS, jakie możemy wyszperać. Generalnie, jeśli nie potrzebujemy zdolności przekierowania I/O dostarczanych przez MS-DOS, odczytujemy wejście klawiatury używając funkcji BIOS dostarczających dużo większej elastyczności.

Wywołujemy usługi klawiaturowe MS-DOS używając instrukcji int 16. BIOS dostarcza następujących funkcji klawiaturowych:

Funkcja # (AH)	Parametry wejściowe	Parametry wyjściowe	Opis
0		al. –znak ASCII ah – kod klawiaturowy	Odczyt znaku. Odczytuje kolejne dostępne znaki z systemowego bufora roboczego. Czeka na naciśnięcie klawisza jeśli bufor jest pusty.
1		ZF – ustawiona jeśli brak klawisza ZF – wyzerowana jeśli klawisz jest dostępny al – kod ASCII ah – kod klawiaturowy	Sprawdza czy znak jest dostępny w buforze roboczym. Ustawia flagę zera jeśli klawisz nie jest dostępny, czyści tą flagę jeśli jest dostępny. Jeśli klawisz jest dostępny, funkcja zwraca kod ASCII i klawiaturowy w ax.

			Wartość ax jest niezdefiniowana jeśli żaden klawisz nie jest dostępny
2		al – flagi przesunięcia	Zwraca bieżący stan flagi przesunięcia w al. Flaga przesunięcia jest zdefiniowana jak następuje: bit 7: przełączony Insert bit 6: przełączony Capslock bit 5: przełączony Numlock bit 4: przełączony Scroll lock bit 3: klawisz Alt naciśnięty bit 2: klawisz Ctrl naciśnięty bit 1: nacisnięty lewy Shift bit 0: naciśnięty prawy Shift
3	al =5 bh = 0, 1,2, 3 dla opóźnienia 1/4 , ½, ¾ sekundy bl =0..1Fh dla 30 /sek do 2/sek		Ustawienia częstotliwości auto powtarzania. Rejestr bh zawiera ilość czasu oczekiwania przed startem operacji auto powtarzania, rejestr bl zawiera częstotliwość auto powtarzania
5	ch = kod klawiaturowy cl = kod ASCII		Przechowuje kod klawisza w buforze. Funkcja ta przechowuje wartość w rejestrze cx na końcu bufora roboczego. Zauważmy, że kod klawiaturowy w ch nie musi odpowiadać kodowi ASCII pojawiającego się w cl/. Ten podprogram będzie po prostu wprowadzał dane jakie dostarczymy do systemowego bufora roboczego
10h		al – znak ASCII ah – kod klawiaturowy	Odczytuje rozszerzony znak. Podobnie jak wywołanie ah =0,poza tym jednym przekazaniem wszystkich kodów klawiszy, ah =0 odrzuca kody, które nie są kompatybilne z PC/XT
11h		ZF –ustawiona jeśli brak klawisza ZF – wyczyszczona jeśli klawisz jest dostępny al – kod ASCII ah – kod klawiaturowy	Jak funkcja ah =0 poza tym jednym nie wyrzuca kod ów klawiszy, które nie są kompatybilne z PC/XT (tzn. znalezione dodatkowe klawisze na klawiaturze 101 klawiszowej)
12h		al – flagi przesunięcia ah – rozszerzone flagi przesunięcia	Zwraca bieżący stan flag przesunięcia w ax. Flagi przesunięcia są zdefiniowane tak: bit 15: naciśnięty klawisz SysReq bit 14: aktualnie naciśnięty Capslock bit 13: aktualnie wciśnięty klawisz Numlock bit 12: aktualnie wciśnięty Scroll lock bit 11: wciśnięty prawy alt bit 10: wciśnięty prawy ctrl bit 9: wciśnięty lewy alt bit 8: wciśnięty lewy ctrl bit 7: przełączony Insert bit 6: przełączony Capslock bit 5; przełączony Numlock bit 4: przełączony Scroll lock bit 3: jakiś alt wciśnięty (pewne maszyny tylko lewy) bit 2: jakiś ctrl wciśnięty bit 1: lewy shift wciśnięty bit 0: prawy shift wciśnięty

Zauważmy, że wiele z tych funkcji nie jest wspartych w każdym BIOS'ie, jaki był napisany. Faktycznie, tylko pierwsze trzy funkcje były dostępne na oryginalnym PC. Jednakże, od kiedy nadszedł AT, większość BIOS'ów wsparło przynajmniej powyższe funkcje. Wiele BIOS'ów dostarcza dodatkowych funkcji, i

jest wiele aplikacji TSR, które mogą rozszerzyć tą listę w przyszłości ,Możemy t łatwo rozszerzyć jeśli mamy takie życzenie.

```
; INT16.ASM
;
; Krótki bierny TSR, który zamienia obsługę int 16h BIOS'a. Podprogram ten demonstruje funkcjonowanie
; każdej z funkcji int 16h, których standardowo dostarcza BIOS
;
; Zauważmy ,że kod ten nie aktualizuje int 2Fh (przerwanie równoczesnych procesów), ani też nie możemy
; usunąć tego kodu z pamięci za wyjątkiem przeładowania  systemu. Jeśli chcemy móc zrobić te dwie rzeczy (jak
; również sprawdzić poprzednią instalację), spójrzmy do rozdziału o programach rezydentnych. Kod taki był
; pominięty dla tego programu z powodu ograniczenia długości.
;
;
; cseg i EndResident muszą wystąpić przed segmentem biblioteki standardowej!

cseg            segemnt para public 'code'
cseg            ends

; Oznaczamy segment, znajdując koniec sekcji rezydentnej

EndResident     segment para public 'Resident'
EndResident     ends

                .xlist
                .include        stdlib.a
                .includelib     stdlib.lib
                .list

byp             equ     < byte ptr >

cseg            segemnt para public 'code'
                assume  cs:cseg, ds:cseg

OldInt16        dword   ?

; Zmienne BIOS:

KbdFlags1       equ     <ds:[17h]>
KbdFlags2       equ     <ds:[18h]>
AltKpd          equ     <ds:[19h]>
HeadPtr         equ     <ds:[1ah]>
TailPtr         equ     <ds:[1ch]>
Buffer          equ     1eh
EndBuf          equ     3eh

KbdFlags3       equ     <ds:[96h]>
KbdFlags4       equ     <ds:[97h]>

incptr          macro   which
                local   NoWrap
                add     bx, 2
                cmp     bx, EndBuf
                jb      NoWrap
                mov     bx, Buffer
NoWrap:         mov     which, bx
                endm
; MyInt16 -     podprogram ten przetwarza żądane funkcje 16h
;               AH      Opis
;               ----    ------------------------------------------------------------------------------------------
```

```
;               00h     Pobiera klawisz z klawiatury, zwraca kod w AX
;               01h     Test dla dostępnego klawisza, ZF=1 jeśli brak, ZF=0 a AX zawiera kolejny kod
;                       klawisza jeśli klawisz jest dostępny
;               02h     pobiera stan przesunięcia. Zwraca stan klawisza Shift w AL.
;               03h     Ustawia częstotliwość auto powtarzania. BH=0,1,2,3 (czas opóźnienia
;                       w czwartej sekundzie), BL=0..1Fh dla 30 znaków/sek do 2 znaków / sek
;                       częstotliwości powtarzania.
;               05h     Przechowuje kod klawiaturowy (w CX) w buforze roboczym
;               10h     pobranie klawisza (to samo co 00h w tej implementacji)
;               11h     Test klawisza (to samo co 01h)
;               12h     Pobranie stanu klawisza rozszerzonego. Zwraca stan w AX

MyInt16         proc    far
                test    ah, 0EFh                ;sprawdzenie od 0h do 10h
                je      GetKey
                cmp     ah, 2                   ;sprawdzenie od 01h do 02h
                jb      TestKey
                je      GetStatus
                cmp     ah, 3                   ;sprawdzenie funkcji Autopowtarzania
                je      SetAutoRpt
                cmp     ah, 5                   ;sprawdzenie funkcji StoreKey
                je      StoreKey
                cmp     ah, 11h                 ; test rozszerzonego opcodu klawisza
                je      TestKey
                cmp     ah, 12h                 ; funkcja rozszerzonego stanu
                je      ExtStatus

; Cóż, jeśli jest funkcja o której nie wiemy , wracamy do kodu wywołującego

                iret

; Jeśli użytkownik określił ah =0 lub ah = 10h, spadamy tutaj (nie będziemy rozróżniać między funkcją getc
; oryginalną a rozszerzoną

GetKey:         mov     ah, 11h
                int     16h                     ;zobaczmy czy klawisz jest dostępny
                je      GetKey                  ;czekamy na naciśnięcie

                push    ds.
                push    bx
                mov     ax, 40h
                mov     ds., ax
                cli                             ;region krytyczny, wyłączamy przerwania
                mov     bx, HeadPtr             ;wskaźnik do kolejnego znaku
                mov     ax, [bx]                ; pobranie znaku
                incptr  HeadPtr
                pop     bx
                pop     ds.
                iret                            ; przywracamy flag przerwania
; TestKey-      Sprawdza czy klawisz jest dostępny w buforze klawiatury. Tu musimy włączyć przerwania (
;               wiec ISR klawiaturowy może umieścić znak w buforze). Generalnie, będziemy chcieli
;               zachować tu flagę przerwań . Ale BIOS zawsze wymusza włączenie przerwa, więc musi być
;               jakieś programy zewnętrzne, które zależą od tego, więc nie „rozwiążemy” tego problemu
;
;               Zwracamy status klawisza w ZF i AX. Jeśli ZF = 1 wtedy żaden klawisz nie jest dostępny a
;               wartość w AX jest nieokreślona. Jeśli ZF=0 wtedy klawisz jest dostępny a AX zawiera kod
;               klawiaturowy / ASCII kolejnego dostępnego klawisza. Ta funkcja nie usuwa kolejnego znaku
;               z bufora wejściowego

TestKey:        sti                             ;włączamy przerwania
```

```
                push    ds
                push    bx
                mov     ax, 40h
                mov     ds., ax
                cli                             ;region krytyczny, wyłączamy przerwania
                mov     bx, HeadPtr
                mov     ax, [bx]                ;BIOS zwraca dostępny kod klawisza
                cmp     bx, TailPtr             ;ZF =1 jeśli pusty bufor
                pop     bx
                pop     ds.
                sti                             ;ponownie włączamy przerwania
                retf    2                       ;zdejmujemy flagi, (ważne  jest ZF )!

; Funkcja GetStatus zwraca zmienną KbdFlags1 w AL.

GetStatus:      push    ds.
                mov     ax, 40h
                mov     ds, ax
                mov     al, KbdFlags1
                pop     ds
                iret

; StoreKey-     Wprowadza wartość w CX do bufora roboczego

StoreKey:       push    ds.
                push    bx
                mov     ax, 40h
                mov     ds, ax
                cli                             ;wyłączamy przerwania, region krytyczny
                mov     bx, TailPtr             ;adres gdzie możemy włożyć  kolejny kod
                push    bx                      ;klawisza
                mov     [bx], cx                ;przechowanie kodu klucza
                incptr  TailPtr                 ;przesuwamy na kolejne wejście w  buforze
                cmp     bx, HeadPtr             ; przepełnienie danych/
                jne     StoreOkay               ;jeśli nie, skok, jeśli tak ignorujemy wejście
                pop     TailPtr                 ;klawisza
                sub     sp, 2                   ;stos dopasowuje ścieżkę alt
StoreOkay:      add     sp, 2                   ;usuwamy śmiecie ze stosu
                pop     bx
                pop     ds.
                iret                            ;przywracamy przerwania

; ExtStatus-    wyszukujemy rozszerzony status klawiatury i zwracamy go w AH, również  zwracamy
;               standardowy status klawiatury w AL.

ExtStatus:      push    ds.
                mov     ax, 40h
                mov     ds, ax
                mov     ah, KbdFlags2
                and     ah, 7Fh                 ;czyścimy końcowe pole sysreq
                test    ah, 100b                ;test bieżącego bitu sysreq
                je      NoSysReq                ;przeskok jeśli zero
                or      ah, 80h                 ;ustawienie końcowego bitu sysreq
NoSysReq:
                and     ah, 0F0h                ;zerowanie bitów alt / ctrl
                mov     al., KbdFlags3
                and     al., 1100b              ;przechwycenie bitów prawych alt / ctrl
                or      ah, al.                 ; dzielimy w AH
                mov     al., KbdFlags2
                and     al., 11b                ;przechwycenie bitów lewego alt / ctrl
```

```
                or      ah , al                         ;dzielimy w AH

                mov     al., KbdFlags1                  ;AL zawiera zwykłe flagi
                pop     ds.
                iret

; SetAutoRpt-   Ustawia częstotliwość autopowtarzania. Na wejściu, bh =0 ,1,2 lub 3 (opóźnienie
;               ¼ sek przed startem autopowtarzania) i bl = 0..1Fh  ( częstotliwość powtarzania
;               od 2: 1 do 30:1 (znak :sekunda).

SetAutoRpt:     push    cx
                push    bx

                mov     al., 0Adh                       ;blokujemy klawiaturę
                call    SetCmd

                and     bh, 11b                 ;wymuszamy właściwą częstotliwość
                mov     cl, 5
                shl     bh, cl                  ;przesuwamy na końcową pozycję
                and     bl, 1Fh                 ;wymuszenie właściwej częstotliwości
                or      bh, bl                  ;  dane bajtu polecenia 8042
                mov     al., 0F3h               ;polecenie ustawienia częstotliwości powtarzania 8042
                call    SendCmd                 ;wysłanie parametrów do 8042

                mov     al., 0AEh               ; odblokowanie klawiatury
                call    SetCmd
                mov     al., 0F4h               ;restart skanowania klawiatury
                call    SendCmd

                pop     bx
                pop     cx
                iret

MyInt16         endp

; SetCmd-       wysyła bajt poleceń w rejestrze AL. do chipu mikrokontrolera klawiatury 8042
;               (rejestr poleceń przy porcie 64h)

SetCmd          proc    near
                push    cx
                push    ax                              ;zachowanie wartości polecenia
                cli                                     ;region krytyczny, przerwań brak

; Czekamy dopóki 8042 nie przetworzy bieżącego polecenia

                xor     cx, cx
Wait4Empty:     in      al, 64h                         ;odczyt rejestru statusu klawiatury
                test    al, 10b                         ;bufor wejściowy pełny?
                loopnz  Wait4Empty                      ;jeśli tak, czekamy dopóki nie będzie pusty

; Okay, wysyłamy polecenie do 8042:

                pop     ax                              ;wyszukanie polecenia
                out     64h, AL.
                sti                                     ; przywracamy przerwania
                pop     cx
                ret
SetCmd          endp

; SendCmd-      Poniższy podprogram wysyła polecenie lub bajt danych do portu danych klawiatury
```

```
;               (port 60h)

SendCmd         proc    near
                push    ds.
                push    bx
                push    cx
                mov     cx, 40h
                mov     ds., cx
                mov     bx, ax                  ;zachowanie bajtu danych

                mov     bh, 3                   ;powtarzamy polecenie
RetryLp:        cli                             ;blokujemy przerwania

; czyścimy flagi błędu, potwierdzenia odbioru i ponownego wysłania w KbdFlags4

                and     byte ptr KbdFlags4, 4fh

;Czekamy dopóki 8042 nie przetworzy bieżącego polecenia

                xor     cx, cx
Wait4Empty:     in      al, 64h                 ;odczyt rejestru statusu klawiatury
                test    al, 10b                 ;bufor wejściowy pełny?
                loopnz  Wait4Empty              ;jeśli tak, czekamy dopóki nie będzie pusty

;Okay, wysyłamy dane do portu 60h

                mov     al., bl
                out     60h, al.
                sti                             ;zezwalamy na przerwania

;Czekamy na nadejście potwierdzenia z ISR'a klawiaturowego:

                xor     cx, cx                  ;czekamy dłuższy czas jeśli trzeba
Wait4Ack:       test    byp  KbdFlags4, 10      ; bit potwierdzenia odbioru
                jnz     GotAck
                loop    Wait4Ack
                dec     bh                      ; robimy powtórkę z nim
                jne     RetryLp

;Jeśli operacja zakończyła się niepowodzeniem po 3 wyszukiwaniach, ustawiamy bit błędu i
; wychodzimy

                or      byp  KbdFlags4, 80h     ;ustawiamy bit błędu

GotAck:         pop     cx
                pop     bx
                pop     ds
                ret
SendCmp         endp

Main            proc

                mov     ax, cseg
                mov     ds, ax

                print
                byte    "INT 16h Replecement", cr, lf
                byte    "Installing…",cr,lf, 0

;Aktualizujemy  wektory przerwań INT 9  i INT 16. Zauważmy, że powyższe instrukcje czynią z
```

```
; cseg aktualny segment danych. Więc możemy tam przechować stare wartości INT 9 i INT 16
; bezpośrednio w zmiennych OldInt9 i OldInt16.

                cli                                             ;wyłączamy przerwania
                mov     ax, 0
                mov     es, ax
                mov     ax, es:[16h*4]
                mov     word  ptr OldInt16, ax
                mov     ax, es:[16*4+2]
                mov     word  ptr OldInt16+2, ax
                mov     es:[16h*4], offset MyInt16
                mov     es:[16h*4+2], cs
                sti                                             ;OK włączamy przerwania

; Jedyna rzecz jaka nam pozostaje to zakończyć i pozostać w pamięci

                print
                byte    „Installed”,cr,lf,0

                mov     ah, 62h                 ;pobieramy wartość PSP programu
                int     21h
                mov     dx, EndResident         ;obliczamy rozmiar programu
                sub     dx, bx
                mov     ax, 3100h               ;polecenie DOS’a TSR
                int     21h
Main            endp
cseg            ends

sseg            segemnt para stack ‘stack’
stk             db      1024  dup (“stack”)
sseg            ends
zzzzzzseg       segemnt para public ‘zzzzzz’
LastBytes       db      16  dup (?)
zzzzzzseg       ends
                end     Main
```

20.5 PODPROGRAM OBSŁUGI PRZERWAŃ KLAWIATUROWYCH

ISR int 16h sprzęga aplikację z klawiaturą. W podobny tonie, ISR int 9 sprzęga między sprzętem klawiatury a ISR’em int 16h. Pracą ISR’a int 9 jest przetwarzanie przerwań sprzętu klawiaturowego, konwersji nadchodzących kodów klawiaturowych do kombinacji kodów klawiaturowego / ASCII i umieszcza je w buforze roboczym, i przetwarza inne wiadomości generowane przez klawiaturę.

Konwertując kody klawiaturowe do kodów klawiaturowych / ASCII, ISR int 9 musi śledzić bieżący stan klawiszy modyfikujących. Kiedy nadchodzi kod klawiaturowy, ISR int 9 może użyć instrukcji xlat do translacji kodu klawiaturowego na kod ASCII używając tablicy int 9 wybranej na podstawie flag modyfikatorów. Inną ważną kwestią jest to ,że program obsługi int 9 musi obsłużyć specjalną sekwencję klawiszy taką jak ctrl-alt-del (reset) i PrtSc. Poniższy kod asemblerowy dostarcza prostego programu obsługi int 9 dla klawiatury. Nie wspiera alt-blok klawiszy kodu ASCII na wejściu lub kilka innych drobnych cech, ale wspiera prawie wszystko co potrzeba programowi obsługi przerwań klawiaturowych. Z pewnością demonstruje wszystkie te techniki jakie musimy znać oprogramowując klawiaturę

```
; INT9.ASM
;
; Krótki TSR dostarczający sterownika dla sprzętowego przerwania klawiatury
;
; Zauważmy, ze kod ten nie aktualizuje int 2Fh (przerwanie równoczesnych procesów), nie możemy usunąć tego
; kodu z pamięci z wyjątkiem ponownego startu. Jeśli chcemy móc zrobić te dwie rzeczy (jak również sprawdzić
; poprzednią instalację), zobaczmy rozdział o programach rezydentnych. Kod taki pominie my w tym programie
; z powodu  ograniczenia długości.
;
```

```
; cseg i EndResident muszą wystąpić przed segmentem biblioteki standardowej!

cseg            segment para public 'code'
OldInt9         dword   ?
cseg            ends

;Oznaczamy segment, znajdując koniec sekcji rezydentnej

EndResident     segment para public 'Resident'
EndResident     ends

                .xlist
                include         stdlib.a
                inlcudelib      stdlib.lib
                .list

NumLockScan     equ     45h
ScrlLockScan    equ     46h
CapsLockScan    equ     3ah
CtrlScan        equ     1dh
AltScan         equ     38h
RShiftScan      equ     36h
LShiftScan      equ     2ah
InsScanCode     equ     52h
DelScanCode     equ     53h

; Bity dla różnych klawiszy modyfikujących

RshfBit         equ     1
LShfBit         equ     2
CtrlBit         equ     4
AltBit          equ     8
SLBit           equ     10h
NLBit           equ     20h
CLBit           equ     40h
InsBit          equ     80h

KbdFlags        equ     <byte ptr ds:[17h]>
KbdFlags2       equ     <byte ptr ds:[18h]>
KbdFlags3       equ     <byte ptr ds:[96h]>
KbdFlags4       equ     <byte ptr  ds:[97]>

byp             equ     < byte ptr>

cseg            segment para public 'code'
                assume  ds: nothing

; Tablica translacji kodów klawiaturowych. Przychodzące z klawiatury kody klawiaturowe stanowią wiersz.
; Kolumny stanowią status modyfikatorów. Słowo na  ich przecięciu to kod klawiaturowy / ASCII do włożenia
; do bufora roboczego PC. Jeśli wartość pobrana z tablicy to zero, wtedy nie kładziemy żadnego znaku do bufora
;
;                       norm    shft    ctrl    alt     num     caps    shcap   shnum

ScanXlat        word    0000h,  0000h,  0000h,  0000h,  0000h,  0000h,  0000h,  0000h
                word    011bh,  011bh,  011bh,  011bh,  011bh,  011bh,  011bh   011bh          ; ESC
                word    0231h,  0231h,  0000h   7800h,  0231h,  0231h,  0231h,  0321h          ;1  !
                word    0332h,  0340h,  0300h,  7900h,  0332h,  0332h,  0332h,  0332h          ;2  @
                word    0433h,  0423h,  0000h,  7a00h,  0433h,  0433h,  0423h,  0423h          ;3  #
                word    0534h,  0524h,  0000h   7b00h,  0534h,  0534h,  0524h,  0524h          ;4  $
                word    0635h,  0625h,  0000h,  7c00h,  0635h,  0635h,  0625h,  0625h          ;5  %
```

```
        word    0736h,  075eh,  071eh,  7d00h,  0736h,  0736h,  075eh,  075eh          ;6 ^
        word    0837h,  0826h,  0000h,  7e00h,  0873h,  0873h,  0826h,  0826h          ;7 &
        word    0938h,  092ah,  0000h,  7f00h,  0938h,  0938h,  092ah,  092ah          ;8 *
        word    0a39h,  0a28h,  0000h,  8000h,  0a39h,  0a39h,  0a28h,  0a28h          ;9 (
        word    0b30h,  0b29h,  0000h,  8100h,  0b30h,  0b30h,  0n29h,  0b29h          ;0 )
        word    0c2dh,  0c5fh,  0000h,  8200h,  0c2dh,  0c2dh,  0c5fh,  0c5fh          ; - _
        word    0d3dh,  0d2bh,  0000h,  8300h,  0d3dh,  0d3dh,  0d2bh,  0d2bh          ;= +
        word    0e08h,  0e08h,  0e7fh,  0000h,  0e08h,  0e08h,  0e08h,  0e08h          ;bksp
        word    0f09h,  0f00h,  0000h,  0000h   0f09h,  0f09h,  0f00h,  0f00h          ; Tab

;               norm    shft    ctrl    alt     num     caps    shcap   shnum

        word    1071h,  1051h,  1011h,  1000h,  1071h,  1051h,  1051h,  1071h          ;Q
        word    1177h,  1057h,  1017h,  1100h,  1077h,  1057h,  1057h,  1077h          ;W
        word    1265h,  1245h,  1205h,  1200h,  1265h,  1245h,  1245h,  1265h          ;E
        word    1372h,  1352h,  1312h,  1300h,  1272h,  1252h,  1252h,  1272h          ;R
        word    1474h,  1454h,  1414h,  1400h,  1474h,  1454h,  1454h,  1474h          ;T
        word    1579h,  1559h,  1519h,  1500h,  1579h,  1559h,  1579h,  1559h          ;Y
        word    1675h,  1655h,  1615h,  1600h,  1675h,  1655h,  1675h,  1655h          ;U
        word    1769h,  1749h,  1709h,  1700h,  1769h,  1749h,  1769h,  1749h          ;I

        word    186fh,  184fh,  180fh,  1800h   186fh,  184fh,  186fh,  184fh          ;O
        word    1970h,  1950h,  1910h,  1900h,  1970h,  1950h,  1970h,  1950h          ;P
        word    1a5bh,  1a7bh,  1a1bh,  0000h,  1a5bh,  1a5bh,  1a7bh,  1a7bh          ; [ {
        word    1b5dh,  1b7dh,  1b1dh,  0000h,  1b5dh,  1b5dh,  1b7dh,  1b7dh          ; ] }
        word    1c0dh,  1c0dh,  1c0ah,  0000h,  1c0dh,  1c0dh,  1coah,  1c0ah          ; enter
        word    1d00h,  1d00h,  1d00h,  1d00h,  1d00h,  1d00h,  1d00h,  1d00h          ; ctrl
        word    1e61h,  1e41h,  1e01h,  1e00h,  1e61h,  1e41h,  1e61h,  1e41h          ; A
        word    1f73h,  1h53h,  1f13h,  1f00h,  1f73h,  1f53h,  1f73h,  1f53h          ; S

;               norm    shft    ctrl    alt     num     caps    shcap   shnum
        word    2064h,  2044h,  2004h,  2000h,  2064h,  2044h,  2064h,  2044h          ; D
        word    2166h,  2146h,  2106h,  2100h,  2166h,  2146h,  2166h,  2146h          ; F
        word    2267h,  2247h,  2207h,  2200h,  2267h,  2247h,  2267h   2247h          ; G
        word    2368h,  2348h,  2308h,  2300h,  2368h,  2348h,  2368h,  2348h          ; H
        word    246ah,  244ah,  240ah,  2400h,  246ah,  244ah,  246ah,  244ah          ; J
        word    256bh,  254bh,  250bh,  2500h,  256bh,  254bh,  256bh,  254bh          ; K
        word    266ch,  264ch,  260ch,  2600h,  266ch,  264ch,  266ch,  264ch          ; L
        word    273bh,  273bh,  0000h,  0000h,  273bh,  273bh,  273ah  273ah          ; ; :
        word    2827h,  2822h,  0000h,  0000h,  2827h,  2827h,  2822h,  2822h          ; ‘ “
        word    2960h,  297eh,  0000h,  0000h,  2960h,  2960h,  297eh,  297eh          ; ` ~
        word    2a00h,  2a00h,  2a00h,  2a00h,  2a00h,  2a00h,  2a00h,  2a00h          ; LShf
        word    2b5ch,  2b7ch,  2b1ch,  0000h,  2b5ch,  2b5ch,  2b7ch,  2b7ch          ; \ |
        word    2c7ah,  2c5ah,  2c1ah,  2c00h,  2c7ah,  2c5ah,  2c7ah,  2c5ah          ; Z
        word    2d78h,  2d58h,  2d18h,  2d00h,  2d78h,  2d58h,  2d78h,  2d58h          ; X
        word    2e63h,  2e43h,  2e03h,  2e00h,  2e63h,  2e43h,  2e63h,  2e43h          ; C
        word    2f76h,  2f56h,  2f16h,  2f00h,  2f76h,  2f56h,  2f76h,  2f56h          ; V

;               norm    shft    ctrl    alt     num     caps    shcap   shnum
        word    3062h,  3042h,  3002h,  3000h,  3062h,  3042h,  3062h,  3042h          ; B
        word    316eh,  3143h,  310eh,  3100h,  316eh,  314eh,  316eh,  314eh          ; N
        word    326dh,  324dh,  320dh,  3200h,  326dh,  324dh,  326dh,  324dh          ; M
        word    332ch,  333ch,  0000h,  0000h,  332ch,  332ch,  333ch,  333ch          ; ,<
        word    342eh,  343eh,  0000h,  0000h,  342eh,  342eh,  343eh,  343eh          ; . >
        word    352fh,  353fh,  0000h,  0000h,  352fh,  352fh,  353fh,  353fh          ; / ?
        word    3600h,  3600h,  3600h,  3600h,  3600h,  3600h,  3600,   3600          ; rshf
        word    372ah,  0000h,  3710h,  0000h,  372ah,  372ah,  0000h,  0000h          ;* PS
        word    3800h,  3800h,  3800h,  3800h,  3800h,  3800h,  3800h,  3800h          ; alt
        word    3920h,  3920h,  3920h,  0000h,  3920h,  3920h,  3920h,  3920h          ; spc
        word    3a00h,  3a00h,  3a00h,  3a00h,  3a00h,  3a00h,  3a00h,  3a00          ; caps
```

```
                word    3b00h,  5400h,  5e00h,  6800h,  3b00h,  3b00h,  5400h,  5400h          ; F1
                word    3c00h,  5500h,  5f00h,  6900h,  3c00h,  3c00h,  5500h,  5500h          ; F2
                word    3d00h,  5600h,  6000h,  6a00h,  3d00h,  3d00h,  5600h,  5600h          ; F3
                word    3e00h,  5700h,  6100h,  6b00h,  3e00h,  3e00h,  5700h,  5700h          ; F4
                word    3f00h,  5800h,  6200h,  6c00h,  3f00h,  3f00h,  5800h,  5800h          ; F5

;                       norm    shft    ctrl    alt     num     caps    shcap   shnum
                word    4000h,  5900h,  6300h,  6d00h,  4000h,  4000h,  5900h,  5900h          ; F6
                word    4100h,  5a00h,  6400h,  6e00h,  4100h,  4100h,  5a00h,  5a00h          ; F7
                word    4200h,  5b00h,  6500h,  6f00h,  4200h,  4200h,  5b00h,  5b00h          ; F8
                word    4300h,  5c00h,  6600h,  7000h,  4300h,  4300h,  500h,   5c00h          ; F9
                word    4400h,  5d00h,  6700h,  7100h,  4400h,  4400h,  5d00h,  5d00h          ; F10
                word    4500h,  4500h,  4500h,  4500h,  4500h,  4500h,  4500h,  4500h          ; num
                word    4600h,  4600h,  4600h,  4600h,  4600h,  4600h,  4600h,  4600h          ; scrl
                word    4700h,  4737h,  7700h,  0000h,  4737h,  4700h,  4737h,  4700h          ; home
                word    4800h,  4838h,  0000h,  0000h,  4838h,  4800h,  4838h,  4800h          ; up
                word    4900h,  4939h,  8400h,  0000h,  4939h,  4900h,  4939h,  4900h          ; pgup
                word    4a2dh,  4a2dh,  0000h,  0000h,  4a2dh,  4a2dh,  4a2dh,  4a2dh          ; -
                word    4b00h,   4b34h, 7300h,  0000h,  4b34h,  4b00h,  4b34h,  4b00h          ; left
                word    4c00h,  4c35h,  0000h,  0000h,  4c35h,  4c00h,  4c35h,  4c00h          ;center
                word    4d00h,  4d36h,  7400h,  0000h,  4d36h,  4d00h,  4d36h,  4d00h          ; right
                word    4e2bh,  4e2bh,  0000h,  0000h,  4e2bh,  4e2bh,  4e2bh,  4e2bh          ; +
                word    4f00h,  4f31h,  7500h,  0000h,  4f31h,  4f00h,  4f31h,  4f00h          ; end

;                       norm    shft    ctrl    alt     num     caps    shcap   shnum
                word    5000h,  5032h,  0000h,  0000h,  5032h,  5000h,  5032h,  5000h          ; down
                word    5100h,  5133h,  7600h,  0000h,  5133h,  5100h,  5133h,  5100h          ; pgdn
                word    5200h,  5230h,  0000h,  0000h,  5230h,  5200h,  5230h,  5200h,         ; ins
                word    5300h,  532eh,  0000h,  0000h,  532eh,  5300h,  532eh,  5300h          ; del
                word    0, 0 ,0 ,0 ,0, 0, 0 ,0                                                 ; --
                word    0, 0 ,0 ,0 ,0, 0, 0 ,0                                                 ; --
                word    0, 0 ,0 ,0 ,0, 0, 0 ,0                                                 ; --
                word    5700h,  0000h,  0000h,  0000h,  5700h,  5700h,  0000h,  0000h          ; F11
                word    5800h,  0000h,  0000h,  0000h,  5800h,  5800h,  0000h,  0000h          ; F12

;*******************************************************************************************
;
; AL zawiera kody klawiaturowe klawiatury

PutInBuffer     proc    near
                push    ds.
                push    bx

                mov     bx, 40h                                 ;ES wskazuje zmienne BIOS
                mov     ds., bx

; Jeśli aktualny kod klawiaturowy to E0 lub E1, musimy odnotować ten fakt, aby poprawnie przetworzyć klawisz
; kursora

                cmp     al., 0e1h
                jne     TryE1
                or      KbdFlags3, 10b                          ;ustawienie flagi E0
                and     KbdFlags3, 0Feh                         ; czyszczenie flagi E1
                jmp     Done

TryE1:          cmp     al., 0e1h
                jne     DoScan
                or      KbdFlags3, 1                            ;ustawienie flagi E1
                and     KbdFlags3, 0Feh                         ; czyszczenie flagi E0
                jmp     Done
```

```
; Zanim zrobimy cokolwiek sprawdzamy czy jest Ctrl-Alt-Del:

DoScan:     cmp     al., DelScanCode
            jnz     TryIns
            mov     bl, KbdFlags
            and     bl, AltBit or CtrlBit           ; Alt = bit 3, ctrl = bit 2
            cmp     bl, AltBit or CtrlBit
            jne     DoPIB
            mov     word  ptr ds:[72h], 1234h       ; flaga gorącego restartu
            jmp     dword  ptr cs:RebootAdrs        ; restart Komputera

RebootAdrs  dword   0ffff0000h                      ; adres resetu

; Sprawdzamy klawisz INS. Jedynka musi przełączyć bit ins we flagach zmiennych klawiaturowych

TryIns:     cmp     al., InsScanCode
            jne     TryInsUp
            or      KbdFlags2, InsBit               ;notujemy czy INS w dole
            jmp     DoPIB                           ; wciśnięty klawisz INS

TryInsUp:   cmp     al., InsScanCode+80h            ; górny kod klawiaturowy INS
            jne     TryShiftDn
            and     KbdFlags2, not InsBit           ; czy INS w górze
            xor     KbdFlags, InsBit                ; przełączamy bit INS
            jmp     QuitPIB

; Obsługujemy tu klawisze lewego i prawego Shift w dole

TryShiftDn: cmp     al., LshiftScan
            jne     TryLShiftUp
            or      KbdFlags, LshiftUp              ;lewy shift w dole
            jmp     QuitPIB

TryLShiftUp: cmp    al, LshiftScan+80h
            jne     TryRShiftDn
            and     KbdFlags, not LShfBit           ; lewy shift w górze
            jmp     QuitPIB

TryRShiftDn: cmp    al, RshiftScan
            jne     TryRShiftUp
            or      KbdFlags, RShfBit               ; prawy shift w dole
            jmp     QuitPIB

TryRShiftUp: cmp    al., RshiftScan+80h
            jne     TryAltDn
            and     KbdFlags, not RshfBit           ; prawy shift w górze
            jmp     QuitPIB
; Obsługa klawisza ALT

TryAltDn:   cmp     al., AltScan
            jne     TryAltUp
            or      KbdFlags, AltBit                ; klawisz alt w dole
GotoQPIB:   jmp     QuitPIB

TryAltUp:   cmp     al, AltScan+80h
            jne     TryCtrlDn
            and     KbdFlags, not AltBit            ; klawisz alt w górze
            jmp     DoPIB
```

```
; Tu działamy z klawiszem control w dole

TryCtrlDn:      cmp     al., CtrlScan
                jne     TryCtrlUp
                or      KbdFlags, CtrlBit               ; klawisz ctrl w dole
                jmp     QuitPIB

TryCtrlUp:      cmp     al, CtrlScan+80h
                jne     TryCapsDn
                and     KbdFlags,  not CyrlBit          ; klawisz ctrl w górze
                jmp     Qi\uitPIB

; tu działamy z klawiszem Capslock w dole

TryCapsDn:      cmp     al., CapsLockScan
                jne     TryCapsUp
                or      KbdFlags2, CLBit                ; Capslock w dole
                xor     KbdFlags, CLBit                 ; przełączenie capslock
                jmp     QuitPIB
TryCapsUp:      cmp     al, CapsLockScan+80h
                jne     TrySLDn
                and     KbdFlags2, not CLBit            ; Capslock w górze
                call    SetLEDs
                jmp     QuitPIB

;działamy z klawiszem Scroll Lock

TrySLDn:        cmp     al., ScrlLockScan
                jne     TrySLUp
                or      KbdFlags2, SLBit                ; scroll lock w doel
                xor     KbdFlags, SLBit                 ; przełączenie scrl lock
                jmp     QuitPIB

TrySLUp:        cmp     al, ScrlLockScan+80h
                jne     TryNLDn
                and     KbdFlags2, not SLBit            ; scrl lock w górze
                call    SetLEDs
                jmp     QuitPIB

; obsługa klawisza NumLock

TryNLDn:        cmp     al., NumLockScn
                jne     TryNLUp
                or      KbdFlags2, NLBit                ;NumLock w dole
                xor     KbdFlags, NLBit                 ; przełączenie numlock
                jmp     QuitPIB

TryNLUp:        cmp     al, NumLockScan+80h
                jne     DoPIB
                and     KbdFlags2, not NLBit              ;numlock w górze
                call    SetLEDs
                jmp     QuitPIB

; Obsługujemy wszystkie inne klawisze:

DoPIB:          test    al., 80h                          ;ignorujemy inne klawisze w górze
                jnz     QuitPIB

; Jeśli najbardziej znaczący bit jest ustawiony w tym punkcie, lepiej będzie mieć zero w AL. W
; przeciwnym razie, jest to górny kod, który możemy bezpiecznie zignorować
```

```
                call    Convert
                test    ax, ax                          ;sprawdzenie na zły kod
                je      QuitPIB

PutCharInBuf:   push    cx
                mov     cx, ax
                mov     ah, 5                           ;przechowanie kodu klawiaturowego w
                int     16h                             ; buforze roboczym
                pop     cx

QuitPIB:        and     KbdFlags3, 0FCh                 ; E0, E1 nie są ostatnim kodem

Done:           pop     bx
                pop     ds.
                ret
PutCharInBuf    endp

;*****************************************************************************************
;
; Convert-      AL zawiera kod klawiaturowy PC. Konwertuje do pary kod ASCII / kod klawiaturowy i
;               zwraca wynik w AX. Kod ten zakłada, że DS. wskazuje przestrzeń zmiennych BIOS (40h)
;

Convert         proc    near
                push    bx

                test    al., 80h                        ;sprawdza czy górny kod
                jz      DownScanCode
                mov     ah, al
                mov     al., 0
                jmp     CSDOne

; Okay, mamy dolny klawisz. Ale przed pójściem dalej, zobaczymy czy nie ma sekwencji ALT- BlokKlawiatury

DownScanCode: mov       bh, 0
                mov     bl, al
                shl     bx,1                            ;mnożymy przez osiem aby obliczyć
                shl     bx, 1                           ; indeks wiersza tablicy xlat kodów
                shl     bx, 1                           ; klawiaturowych

; Obliczamy indeks modyfikatora:
;
;               jeśli alt wtedy modyfikator = 3

                test    KbdFlags, AltBit
                je      NotAlt
                add     bl, 3
                jmp     DoConvert

;               jeśli ctrl, wtedy modyfikator = 2

NotAlt:         test    KbdFlags, CtrlBit
                je      NotCtrl
                add     bl, 2
                jmp     DoConvert

; Bez względu na ustawienie shift, musimy działać z numlock I capslock. Numlock jest tylko problemem jeśli
; kod klawiaturowy jest większy lub równy 47h. Capslock, jeśli ten kod klawiaturowy jest mniejszy niż ten.
```

```
NotCtrl:        cmp     al., 47h
                jb      DoCapsLk
                test    KbdFlags, NLBit                 ;testowanie bitu Numlock
                je      NoNumLck
                test    KbdFlags, LShfBit ot RshfBit    ;sprawdzenie l/p shift
                je      NumOnly
                add     bl, 7
                jmp     DoConvert

NumOnly:        add     bl, 4                           ;tylko numlock
                Jmp     DoConvert

; Jeśli  numlock nie jest aktywny, zobaczymy czy jest klawisz shift

NoNumLck:       test    KbdFlags, LShfBit  or RshfBit   ;sprawdza l/p shift
                je      DoConvert                       ;normalnie jeśli brak shift
                add     bl, 1
                jmp     DoConvert

; Jeśli wartość kodu klawiaturowego jest poniżej 47h, musimy sprawdzić capslock

DoCapsLk:       test    KbdFlags, CLBit                 ;sprawdzenie bitu capslock
                je      DoShift
                test    KbdFlags, LShfBit or RshfBit    ; sprawdzenie l/p shift
                je      CapsOnly
                add     bl, 6                           ;Shift i capslock
                jmp     DoConvert

CapsOnly:       add     bl, 5                           ;CapsLock
                jmp     DoConvert

;Cóż, nic więcej nie jest aktywne, sprawdzamy klawisz shift

DoShift:        test    KbdFlags, LShfBIt  ot RshfBit   ; l/ p shift
                je      DoConvert
                add     bc, 1                           ;Shift
DoConvert:      shl     bx, 1                           ; tablica słó
                mov     ax, ScanXlat[bx]
CSDOne:         pop     bx
                Ret
Convert         endp

; SetCmd-       Wysyła bajt poleceń w rejestrze AL do  chipu mikrokontrolera klawiatury 8042
;               (rejestr poleceń przy porcie 64h)
;

SetCmd          proc    near
                push    cx
                push    ax                      ;zachowujemy wartość polecenia
                cli                             ;region krytyczny, żadnych przerwań

; Czekamy dopóki 8042 przetworzy bieżące polecenie

                xor     cx, cx
Wait4Empty:     in      al, 64h                 ;rejestr odczytu stanu klawiatury
                test    al., 10b                ;pełny bufor?
                loopnz  Wait4Empty              ;jeśli tak czekamy aż będzie pusty

;Okay, wysyłamy polecenie do 8042:
```

```
                pop     ax                      ;wyszukujemy polecenie
                out     64h, al
                sti                             ;włączenie przerwań
                pop     cx
                ret
SetCmd          endp

; SendCmd-      Poniższy podprogram wysyła polecenie lub bajt danych do portu danych klawiatury
;               (port 60h)

SendCmd         proc    near
                push    ds.
                push    bx
                push    cx
                mov     cx, 40h
                mov     ds., cx
                mov     bx, ax                          ;zachowanie bajtu danych

                mov     bh, 3                           ;ponowienie polecenia
RetryLp:        cli                                     ; blokujemy przerwania

; Flagi czyszczenia błędu, potwierdzenie odebrania i ponownego wysłani w KbdFlags4

                and     byte  ptr KbdFlags4, 4fh

; czekamy dopóki 8042 nie przetworzy bieżącego polecenia

                xor     cx, cx
Wait4Empty:     in      al, 64h                 ;rejestr odczytu stanu klawiatury
                test    al., 10b                ;pełny bufor?
                loopnz  Wait4Empty              ;jeśli tak czekamy aż będzie pusty

; Okay wysyłamy daną do portu 60h

                mov     al., bl
                out     60h, al.
                sti                             ;włączamy przerwania

; Czekamy na nadejście potwierdzenia  z ISR'a klawiaturowego:

                xor     cx, cx                  ;czekamy dłuższy czas jeśli trzeba
Wait4Ack:       test    byp KbdFlags4, 10h      ;bit potwierdzenia odbioru
                jnz     GotAck
                loop    Wait4Ack
                dec     bh                      ; ponowienie
                jne     RetryLp

; jeśli operacja się nie powiodła po 3 próbach, ustawiamy bit błędu i wychodzimy

                or      byp     KbdFlags4, 80h  ; ustawiony bit błędu

GotAck:         pop     cx
                pop     bx
                pop     ds.
                ret
SendCmd         endp

;SetLEDs-       uaktualnia bity LED KbdFlags4 ze zmiennej KbdFlags a potem przekazuje
;               nowe ustawienie flagi klawiatury
;
```

```
SetLEDs     proc    near
            push    ax
            push    cx
            mov     al., KbdFlags
            mov     cl, 4
            shr     al., cl
            and     al., 111b
            and     KbdFlags4, 0F8h         ;czyszczenie bitów LED
            or      KbdFlags4, al.          , maskowanie nowych bitów
            mov     ah, al.                 ;zachowanie bitów LED

            mov     al., 0ADH               ;zablokowanie klawiatury
            call    SetCmd

            mov     al., 0Edh               ;ustawienie poleceń LED 8042
            call    SendCmd                 ; wysłanie polecenia do 8042
            mov     al., ah                 ; pobranie parametrów bajtu
            call    SendCmd                 ;wysłanie parametrów do 8042

            mov     al., 0AEh               ; odblokowanie klawiatury
            call    SetCmd
            mov     al., 0F4h               ;restart skanowania klawiatury
            call    SendCmd
            pop     cx
            pop     ax
            ret
SetLEDs     endp

; MyInt9-   Podprogram obsługi przerwania dla sprzętowego przerwania klawiatury

MyInt9      proc    far
            push    ds
            push    ax
            push    cx

            mov     ax, 40h
            mov     ds., ax

            mov     al., 0ADh                       ;blokada klawiatury
            call    SetCmd
            cli                                     ;blokada przerwań

            xor     cx, cx
Wait4Data:  in      al, 64h                         ;odczyt stanu portu klawiatury
            test    al., 10b                        ;dana w buforze?
            loopz   Wait4Data                       ;czekaj póki dana jest dostępna
            in      al., 60h                        ;pobrane danej klawiaturowej
            cmp     al., 0EEh                       ; odpowiedź echa?
            je      QuitInt9
            cmp     al., 0FAh                       ; potwierdzenie?
            jne     NotAck
            or      KbdFlags4, 10h                  ;ustawienie bitu potwierdzenia
            jmp     QuitInt9

NotAck:     cmp     al., 0Feh                       ;polecenie ponownego wysłania?
            jne     NotResend
            or      KbdFlags4, 20h                  ; ustawienie bitu ponownego wysłania
            jmp     QuitInt9
```

;Notka: inne polecenia sterownika klawiatury, wszystkie mają dwój najbardziej znaczący bit ustawiony

```
; a podprogram PutInBuffer będzie je ignorował

NotResend:      call    PutInBuffer                     ;włożenie do bufora roboczego
QuitInt9:       mov     al., 0AEh               ;ponowne odblokowanie klawiatury
                call    SetCmd

                mov     al., 20h                ; wysłanie EOI (koniec przerwania)
                out     20h, al.                ; do PIC 8259A
                pop     csx
                pop     ax
                pop     ds
                iret
MyInt9          endp

Main            proc
                assume  ds:cseg

                mov     ax, cseg
                mov     ds, ax

                print
                byte    “INT 9 Replacement” ,cr,lf
                byte    “Installing…”,cr, lf, 0

; Aktualizujemy  wektor przerwania INT 9. Zauważmy, że powyższe instrukcje zrobiły cseg
; bieżącym segmentem danych, wiec możemy przechować starą wartość INT 9bezpośrednio w
; zmiennej OldInt9

                cli                             ;przerwania wyłączone
                mov     ax, 0
                mov     es, ax
                mov     ax, es:[9*4]
                mov     word  ptr OldInt9, ax
                mov     ax, es:[9*4+2]
                mov     word  ptr OldInt9+2, ax
                mov     es:[9*4].  Offset MyInt9
                mov     es:[984+2], cs
                sti                             ; włączamy przerwania
; pozostało nam zakończyć i pozostawić w pamięci

                print
                byte    „Installed”, cr, lf,0

                mov     ah, 62h                 ; pobranie wartości PSP programu
                int     21h
                mov     dx, EndResident         ;obliczamy rozmiar programu
                sub     dx, bx
                mov     ax, 3100h               ;polecenie DOS TSR
                int     21h
Main            endp
cseg            ends

sseg            segment para stack ‘stack’
stk             byte    1024   dup  (“stack”)
sseg            ends

zzzzzzseg       segment para public ‘zzzzzz’
LastBytes       db      16  dup (?)
zzzzzzseg       ends
                end     Main
```

20.6 AKTUALIZOWANIE PODPROGRAMU OBSŁUGI PRZERWANIA INT 9

Dla wielu programów, takich jak programy pop-up lub poprawionej klawiatury, możemy musieć zatrzymać pewne „gorące klawisze" i przekazać wszystkie pozostałe kody klawiaturowe do domyślnego podprogramu obsługi przerwania klawiaturowego. Możemy wprowadzić ISR int 9 do łańcucha przerwań dziewięć podobnie jak każde inne przerwanie. Kiedy klawiatura przerywa systemowi, wysyła kod klawiaturowy, program obsługi przerwania może odczytać ten kod z portu 60h i zadecydować czy przetwarzać sam kod klawiaturowy czy przekazać sterowanie do innego programu obsługi int 9. Poniższy program demonstruje tą zasadę; deaktywuje funkcję resetu ctrl-alt-del na klawiaturze poprzez przechwycenie i odrzucenie usuniętych kodów klawiaturowych kiedy bity ctrl i alt są ustawione w bajcie flagi klawiatury

```
; NORESET.ASM
;
; Krótki TSR, który aktualizuje przerwanie int 9 i przechwytuje sekwencje klawiszy ctrl-alt-del.
;
; Zauważmy, że kod ten nie aktualizuje przerwania 2Fh (przerwania równoczesnych procesów), nie można
; usunąć go z pamięci z wyjątkiem restartu. Jeśli chcemy móc zrobić te dwie rzeczy (jak również sprawdzenie
; poprzedniej instalacji) zajrzymy do rozdziału o programach rezydentnych. Kod taki został pominięty dla tego
; programu z powodu ograniczenia długości.
;
; cseg i EndResident muszą pojawić się przed segmentami biblioteki standardowej!
;
cseg            segment para  public 'code'
OldInt9         dword   ?
cseg            ends

;Oznaczamy segment znajdując koniec sekcji rezydentnej

EndResident     segment para public 'Resident'
Endresident     ends

                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list
DelScanCode     equ     53h

;Bity dla zmiennych klawiszy modyfikujących

CtrlBit         equ     4
AltBit          equ     8
KbdFlags        equ     <byte ptr ds:[17h]>

cseg            segment para  public 'code'
                assume  ds:nothing
;SetCmd-        Wysyła bajt poleceń w rejestrze AL do chipa mikrokontrolera klawiatury 8042

SetCmd          proc    near
                push    cx
                push    ax                      ;zachowujemy wartość polecenia
                cli                             ;region krytyczny, żadnych przerwań

; Czekamy dopóki 8042 nie przetworzy bieżącego polecenia

                xor     cx, cx
Wait4Empty      in      al, 64h                 ;odczyt rejestru stanu klawiatury
                test    al., 10b                ;czy bufor pełny?
                loopnz  Wait4Empty              , jeśli tak, czekamy aż będzie pusty
```

```
;okay, wysyłamy polecenia do 8042:

                pop     ax                      ;wyszukujemy polecenie
                out     64h, al.
                sti                             ;włączamy przerwania
                pop     cx
                ret
SetCmd          endp

; MyInt9-       Podprogram obsługi przerwania dla sprzętowego przerwania klawiatury. Testuje
;               aby sprawdzić czy użytkownik nacisnął klawisz DEL. Jeśli nie, przekazuje
;               sterowanie do oryginalnego programu obsługi int 9. Jeśli tak sprawdza czy
;               są wciśnięte klawisze ctrl i alt; jeśli nie przekazuje sterowanie do oryginalnego programu
;               W przeciwnym razie zjada kody klawiaturowe nie przekazując bezpośrednio DEL .

MyInt9          proc    far
                push    ds.
                push    ax
                push    cx

                mov     ax, 40h
                mov     ds., ax

                mov     al., 0ADh               ;blokada klawiatury
                call    SetCmd
                cli                             ;blokada przerwań
                xor     cx, cx
Wait4Data;      in      al, 64h                 ; odczyt stanu portu klawiatury
                test    al., 10b                ;dana w buforze?
                loopz   Wait4Data               ;czekamy dopóki dana jest dostępna

                in      al., 60h                ;pobranie danej klawiaturowej
                cmp     al., DelScanCode        ; czy to klawisz Delete?
                jne     OrigiInt9
                mov     al, KbdFlags            ;okay. Mamy Del, czy ctrl+alt wciśnięte?
                and     al., AltBit or CtrlBit
                cmp     al, AltBit  or CtrlBit
                jne     OrigInt9

; Jeśli wciśnięto ctrl+alt+del, zjadamy kod DEL I nie przekazujemy go bezpośrednio

                mov     al., 0AEh               ;odblokowanie klawiatury
                call    SetCmd

                mov     al., 20h                ;wysłanie EOI (koniec przerwania)
                out     20h, al.                ;do PIC 8259A
                pop     cx
                pop     ax
                pop     ds
                iret

; Jeśli ctrl i alt nie są oba wciśnięte, przekazujemy DEL do oryginalnego programu obsługi INT 9

OrigInt9:       mov     al., 0Aeh               ;odblokowanie klawiatury
                call    SetCmd

                pop     cx
                pop     ax
                pop     ds.
                jmp     cs:OldInt9
```

```
MyInt9          endp

Main            proc
                assume  ds: cseg

                mov     ax, cseg
                mov     ds, ax

                print
                byte    "Ctrl-Alt-Del Filter",cr, lf
                byte    "Installing…", cr, lf,0

;Aktualizujemy wektor przerwania INT 9. Zauważmy, że powyższe instrukcje uczyniły cseg aktualnym
; segmentem danych, więc możemy przechować starą wartość INT 9 bezpośrednio w zmiennej OldInt9

                cli                             ;wyłączamy przerwania
                mov     ax, 0
                mov     es, ax
                mov     ax, es:[9*4]
                mov     word  ptr OldInt9, ax
                mov     ax, es:[9*4+2], ax
                mov     word  ptr OldInt9+2, ax
                mov     es:[9*4], offset MyInt9
                mov     es:[9*4+2], cs
                sti                             ;włączamy przerwania

; Pozostało zakończyć I pozostawić w pamięci

                print
                byte    „Installed", cr,lf,0
                mov     ah, 62h                 ;pobranie wartości PSP programu
                int     21h

                mov     dx, EndResident         ;obliczanie rozmiaru programu
                sub     dx, bx
                mov     ax, 3100h               ;polecenie DOS TSR
                int     21h
Main            endp
cseg            ends

sseg            segment para stack 'stack'
stk             db      1024   dup ("stack")
sseg            ends

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       db      16 dup (?)
zzzzzzseg       ends
                end     Main
```

20.7 SYMULOWANIE UDERZEŃ W KLAWISZE

Czasami możemy chcieć pisać pogramy, które przekazują naciśnięte klawisze do innej klawiatury. Na przykład możemy chcieć napisać makro klawiaturowe TSR, które pozwoli nam przechwycić pewne klawisze na klawiaturze i wysyłać sekwencję klawiszy bezpośrednio do odpowiedniej aplikacji. Być może będziemy chcieli oprogramować całe ciągi znaków normalnie nie używanych sekwencji klawiaturowej(np. ctrl-up lub ctrl-down). W pewnych przypadkach nasz program będzie używał pewnych technik do przekazania znaków do aplikacji pierwszoplanowej . Są trzy dobrze znane techniki dla zrobienia tego: przechowanie kodu klawiaturowego/ ASCII bezpośrednio w buforze klawiatury, użyć flagi śledzenia dla symulacji instrukcji in al., 60h lub oprogramowanie mikrokontrolera zintegrowanego 8042 do przekazywania nam kodu klawiaturowego. Kolejne trzy sekcje opisują te techniki szczegółowiej.

20.7.1 WSTAWIANIE ZNAKÓW DO BUFORA ROBOCZEGO

Być może najłatwiejszym sposobem wstawiania naciśniętych klawiszy do aplikacji jest wprowadzenie ich bezpośrednio do systemowego bufora roboczego. Większość nowoczesnych BIOS'ów dostarcza w tym celu funkcji int 16h. Nawet jeśli nasz system nie dostarcza tej funkcji łatwo jest napisać swój własny kod wprowadzający dane do systemowego bufora roboczego; lub można skopiować kod z programu int 16h pokazanego wcześniej w tym rozdziale.

Fajne w tym podejściu jest to, ze możemy działać ze znakami ASCII (przynajmniej dla tych sekwencji klawiszy które są ASCII) Nie musimy martwić się o wysyłanie przesunięcia górnego i dolnego kodu dla kodu klawiaturowego „A", aby uzyskać duża literę „A", musimy tylko wprowadzić 1E41h do bufora. Faktycznie większość programów ignoruje kody klawiaturowe, więc możemy po prostu wprowadzić 0041h do bufora i prawie każda aplikacja będzie akceptowała kod klawiaturowy zero.

Główna wadą techniki wkładania do bufora jest to ,że wiele (popularnych) aplikacji omija DOS i BIOS kiedy odczytuje klawiaturę. Program takie wchodzą bezpośrednio do portu klawiatury (60h) i odczytują dane. Jako takie pokazywanie kodów klawiaturowych / ASCII w buforze roboczym nie będzie odnosiło skutku. Idealnie byłoby gdybyśmy mogli wprowadzić kod klawiaturowy bezpośrednio do chipu mikrokontrolera klawiatury i zwracać ten kod klawiaturowy jak gdyby jakiś rzeczywiście naciśnięty klawisz. Niestety ,nie ma uniwersalnego sposobu na zrobienie tego. Jednakże są bliskie przybliżenia.

20.7.2 UŻYWANIE FLAGI ŚLEDZENIA 80X86 DO SYMULOWANIA INSTRUKCJI IN AL., 60H

Jedyny sposób zajęcia się aplikacjami, które uzyskują bezpośrednio dostęp do sprzętu klawiatury to zasymulowanie zbioru instrukcji 80x86. Na przykład przypuśćmy, że przejmiemy sterowanie ISR int 9 i wykonamy każdą instrukcję pod naszą kontrolą. Możemy wybrać zezwolenie na wszystkie instrukcje z wyjątkiem instrukcji in wykonywanej zazwyczaj. Przy napotykaniu instrukcji in (której ISR klawiaturowy używa do odczytu) sprawdzamy czy jest dostęp do portu 60h. Jeśli tak, po prostu ładujemy rejestr al. żądanym kodem klawiaturowym zamiast rzeczywiście wykonywaną instrukcją in. Ważne jest, aby sprawdzić instrukcję out, ponieważ ISR klawiaturowy będzie chciał wysłać sygnał EOI do PIC 8259A po odczytaniu danej klawiaturowej, możemy po prostu zignorować instrukcje out ,która zapisuje do portu 20h.

Jedyną trudniejszą częścią jest powiedzenie 80x86 o przekazaniu sterowania do naszego podprogramu kiedy napotykamy pewne instrukcje (jak in i out) i wykonujemy normalnie inne instrukcje. Nie jest to bezpośrednio możliwe w trybie rzeczywistym, jest to bliskie przybliżenie jakie możemy uczynić. CPU 80x86 dostarczają flagę śledzenia, która generuje wyjątek po wykonaniu każdej instrukcji. Normalnie debuggery używają flagi śledzenia do przejścia przez program w pojedynczych krokach. Jednakże poprzez napisanie własnego programu obsługi wyjątku dla wyjątku śledzenia możemy wzmocnić sterowanie maszyną pomiędzy wykonaniem każdej instrukcji. Wtedy możemy przyjrzeć się opcodowi kolejnej instrukcji do wykonania. Jeśli nie jest to instrukcja in lub out, możemy określić adres I/O i zadecydować czy symulować czy wykonać instrukcję.

Oprócz instrukcji in i out, będziemy musieli zasymulować instrukcję int. Powód jest taki, że instrukcja int odkłada flagi na stos a potem czyści bit śledzenia w rejestrze flag To oznacza, że podprogram obsługi przerwań powiązany z instrukcją int wykonuje się normalnie a my możemy zgubić pojawiające się w tym instrukcje in i out Jednakże, łatwo jest zasymulować instrukcję int pozostawiając włączoną flagę śledzenia, więc dodamy int do naszej listy instrukcji do przetłumaczenia.

Jedyny problem x tym podejściem jest taki, że jest wolne. Chociaż podprogram pułapki śledzenia będzie wykonywał tylko kilka instrukcji na każde wywołanie, robi to dla każdej instrukcji ISR'a int 9. W wyniku, podczas symulacji, podprogram obsługi przerwania będzie działał 10 d0 20 razy wolniej niż kod rzeczywisty. Nie jest to generalnie problem ponieważ większość ISR' ów klawiaturowych jest bardzo krótkich. Jednakże, możemy spotkać się z aplikacją, która ma duży wewnętrzny ISR int 9 a metoda ta zauważalnie zwolni program. Jednak dla większości aplikacji technika ta działa poprawnie i nie zauważymy żadnego spowolnienia wydajności podczas pisania na klawiaturze.

Poniższy kod asemblerowy dostarcza krótkiego przykładu programu obsługi śledzenia, który symuluje naciskanie klawisz w ten sposób:

```
                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list

cseg            segemnt para public 'code'
```

```
                assume  ds:nothing

; ScanCode musi być w segmencie kodu
ScanCode        byte    0

;*****************************************************************************
;
; KbdSim-       Przekazuje kod klawiaturowy w AL. bezpośrednio do kontrolera klawiatury używając flagi
;               śledzenia. Sposób w jaki działa to włączenie bitu śledzenia w rejestrze flag. Każda instrukcja
;               wtedy wywołuje pułapkę śledzenia (Zainstalowany0 program obsługi śledzenia patrzy na
;               każdą instrukcję obsługującą IN, OUT, INT i inne specjalne instrukcje. Po napotkaniu IN AL.,
;               60 (lub odpowiednika) kod ten symuluje tą instrukcję i zwraca określony kod klawiaturowy
;               zamiast aktualnie wykonywanej instrukcji IN. Inne instrukcje również potrzebują specjalnego
;               traktowania. Zobacz kod szczegółowo. Kod ten jest całkiem niezły przy symulowaniu sprzętu
;               ale działa dość wolno i ma kilka problemów kompatybilności/

KbdSim          proc    near

                pushf
                push    es
                push    ax
                push    bx

                xor     bx, bx                  ;wskazuje tablicę wektorów przerwań
                mov     es, bx                  ; (do symulowania INT 9)
                cli                             ; żadnych przerwań
                mov     cs:ScanCode, al.        ; zachowanie wyjściowego kodu
                                                ; klawiaturowego

                push    es:[1*4]                ; zachowanie aktualnego wektora INT 1
                push    es:2[1*4]               ;aby odzyskać go później

;wektor INT 1 wskazuje na nasz program obsługi INT 1:

                mov     word  ptr es:[1*4],  offset  MyInt1
                mov     word  ptr es:[1*4+2],  cs

; Włączamy pułapkę śledzenia (bit 8 rejestru flag)

                pushf
                pop     ax
                or      ah, 1
                push    ax
                popf

;Symulujemy instrukcje INT 9. Notka: nie możemy tu w rzeczywistości wykonać INT 9 ponieważ instrukcje
; INT wyłączają operację śledzenia

                pushf
                call    dword  ptr  es:[9*4]

; Wyłączamy operację śledzenia

                pushf
                pop     ax
                and     ah, 0feh                ;czyścimy bit śledzenia
                push    ax
                popf

; Blokujemy operację śledzenia
```

```
                pop     es:[1*4+2]                      ;przywrócenie poprzedniego programu
                pop     es:[1*4]                        ;obsługi INT 1
; Okay, zrobione. Przywracamy rejestry i wracamy

VMDone:         pop     bx
                pop     ax
                pop     es
                popf
                ret
KbdSim          endp

;-----------------------------------------------------------------------------------------------------------------------
;
; MyInt1- Obsługuje pułapkę śledzenia (INT1). Kod ten przygląda się kolejnemu opcodowi określając czy jest to
;jeden ze specjalnych opcodów, jaki musimy obsłużyć sami.

MyInt1          proc    far
                push    bp
                mov     bp, sp                  ;zyskujemy dostęp do adresu powrotnego poprzez
                push    bx                      ; BP
                push    ds.

; Jeśli jesteśmy tu, to dlatego, że ta pułapka śledzenia  jest bezpośrednio z powodu naszego dziurawego bitu
; śledzenia. Przetwarzamy pułapkę śledzenia dla symulowania zbioru instrukcji 80x86
;
; Pobranie adresu powrotnego w DS.:BX

NextInstr:      lds     bx, 2[bp]

;Poniżej jest specjalny przypadek do szybkiego eliminowania większości opcodów I przyspieszenia tego kodu
; poprzez ograniczenie ilości

                cmp     byte  ptr[bx], 0cdh     ;większość opcodów jest mniejszych niż 0cdh,
                jnb     NotSimple               ; w związku z tym szybko wrócimy do
                pop     ds.                     ;;rzeczywistego programu
                pop     bx
                pop     bp
                iret

NotSimple:      je      IsIntInstr              ; Czy to jest instrukcja INT

                mov     bx, [bx]                ;pobranie opcodu bieżącej instrukcji
                cmp     bl, 0e8h                ; opcod CALL
                je      ExecInstr
                jb      TryInOut0

                cmp     bl, 0ech                ;IN al dx instrukcja
                je      MayBeIn60
                cmp     bl, 0eeh                ;OUT dx,  al instrukcja
                je      MayBeOut20
                pop     ds                      ; zwykła instrukcja
                pop     bx
                pop     bp
                iret

TryInOut0:      cmp     bx, 60e4h               ;IN  al,  instrukcja 60h
                je      IsINAL60
                cmp     bx, 20e6h               ; out 20, al instrukcja
                je      IsOut20
```

```
; Jeśli nie była to jedna z magicznych instrukcji , wykonujemy ja i kontynuujemy

ExecInstr:      pop     ds.
                pop     bx
                pop     bp
                iret

; Jeśli ta instrukcja to IN AL, DX, musimy popatrzyć na wartość w DX odkreślając czy jest to rzeczywiście
; instrukcja IN Al., 60h

MayBeIn60:      cmp     dx, 60h
                jne     ExecInstr
                inc     word  ptr 2[bp]             ;przeskakujemy 1 bajt tej instrukcji
                mov     al., cs:ScanCode
                jmp     NextInstr

;Jeśli jest to instrukcja IN AL, 60h, symulujemy ją poprzez załadowanie bieżącego kodu klawiaturowego do AL.

IsInAL60:       mov     al., cs:ScanCode
                add     word  ptr 2[bp], 2          Przeskakujemy ponad dwoma bajtami instrukcji
                jmp     NextInstr

; Jeśli jest to instrukcja OUT DX, AL., musimy popatrzyć do DX aby zobaczyć czy wychodzimy do lokacji 20h
; (8259)

MayBeOut20:     cmp     dx, 20h
                jne     ExecInstr
                inc     word  ptr 2[bp]             ;przeskakujemy ten 1 bajt instrukcji
                jmp     NextInstr

; Jeśli jest to instrukcja OUT 20h, al., po prostu przeskakujemy to

IsOut20:        add     word  ptr 2[bp], 2          ;przeskakujemy instrukcję
                jmp     NextInstr

; IsIntInstr-       Wykonujemy ten kod jeśli jest to instrukcja INT
;
; Problem z instrukcjami INT jest taki, że resetują bit śledzenia przy wykonaniu. Dla pewnych z nich możemy t
; tego nie mieć
;
;Notka: w tym punkcie stos wygląda jak następuje:
;
;               flags
;
;               rtn cs -+
;                       |
;               rtn ip  +-- Wskazuje kolejną instrukcję CPU do wykonania
;               bp
;               bx
;               ds.
;
;               Musimy zasymulować właściwą instrukcję INT poprzez:
;                       (1) dodanie dwa do adresu powrotnego na stosie (więc wracamy poza instrukcję INT)
;                       (2) odłożenie flag na stos
;                       (3) odłożenie fałszywego adresu powrotu na stos, który symuluje adres powrotu
;                           przerwania INT 1, ale który „zwraca” nam  określony program obsługi przerwania
;
; Wszystkie te wynik na stosie wyglądają jak następuje:
;
```

```
;                   flags
;
;                   rtn cs -+
;                           |
;                   rtn   +-- Wskazuje kolejną instrukcję poza instrukcję INT
;
;                   flags ---  Fałszywe flagi dla symulowania tych odłożonych przez instrukcję INT
;
;                   rtn cs -+
;                           |
;                   rtn ip  +-- „Adres powrotny” który wskazuje na ISR dla tego INT
;                   bp
;                   bx
;                   ds.

IsINTInstr:         add     word  ptr 2][bp], 2     ;wypadniecie adresu powrotnego poza instrukcję INT
                    mov     bl, 1[bx]
                    mov     bh, 0
                    shl     bx, 1                   ;Mnożenie przez 4 do pobrania adresu wektora
                    shl     bx, 1

                    push    [bp-0]                  ;pobranie i zachowanie BP
                    push    [bp-2]                  ;pobranie i zachowanie BX
                    push    [bp-4]                  ; pobranie i zachowanie DS.

                    push    cx
                    xor     cx, cx                  ;wskazuje DS jako tablicę wektorów przerwań
                    mov     ds., cx

                    mov     cx, [bp+6]              ;pobranie oryginalnych flag
                    mov     [bp-0], cx              ;zachowanie  odłożonych flag

                    mov     cx, ds.:2[bx]           ;pobranie wektora I użycie go jako adresu powrotnego
                    mov     [bp-2], cx
                    mov     cx, ds.:[bx]
                    mov     [bp-4] ,cx

                    pop     cx
                    pop     ds
                    pop     bx
                    pop     bp
                    iret

MyInt1              endp

;Program główny – symuluje jakieś naciśnięcia klawiszy dal demonstracji powyższego kodu

Main                proc

                    mov     ax, cseg
                    mov     ds, ax

                    print
                    byte    “Simulating keystrokes via Trace Flag”,cr,lf
                    byte    “This program places ‘DIR’ in the keybord buffer”
                    byte    cr, lf, 0

                    mov     al, 20h                         ; dolny kod “D”
                    call    KbdSim
                    mov     al., 0a0h                       ; górny kod “D”
```

```
                call    KbdSim

                mov     al., 17h                ; dolny kod “I”
                call    KbdSim
                mov     al., 97h                ;górny kod “I”
                call    KbdSim

                mov     al. 13h                 ; dolny kod “R”
                call    KbdSim
                mov     al., 93h                ;górny kod “R”
                call    KbdSim

                mov     al., 1Ch                ;  dolny kod klawisza Enter
                call    KbdSim
                mov     al., 9Ch                ;górny kod klawisza Enter
                call    KbdSim

                ExitPgm
Main            endp

cseg            ends
sseg            segment para stack ‘stack’
stk             byte    1024   dup (“stack”)
sseg            ends

zzzzzzseg       segemnt para public ‘zzzzzz’
LastBytes       db      16   dup (?)
Zzzzzzseg       ends
                end     Main
```

20.7.3 UŻYCIE MIKROKONTROLERA 8042 DO SYMULOWANIA UDERZEŃ W KLAWISZE

Chociaż flaga śledzenia oparta na podprogramie „klawiaturowym” działa z większością oprogramowania, bezpośrednio komunikując się ze sprzętem, ma jeszcze kilka problemów. Ściśle, nie działają pod wszystkimi programami, które operują w trybie chronionym poprzez bibliotekę „DOS Extender” (biblioteka programistyczna, która pozwala programistom na uzyskani dostępu do więcej niż jednego megabajt pamięci podczas pracy pod DOS) Ta ostania technika jakiej się przyjrzymy jest to oprogramowanie zintegrowanego mikrokontrolera klawiatury 8042 dla przekazywani nam uderzeń w klawisze. Są dwa sposoby zrobienia tego: sposób PS/2 i sposób twardy.

Mikrokontroler PS/2 zawiera ściśle zaprojektowane polecenia do zwracania użytkownikowi programowalnych kodów klawiaturowych systemu .Poprzez wpisanie bajtu 0D2h do portu poleceń kontrolera (64h) i bajt kodu klawiaturowego do portu 60h, możemy zmusić kontroler do zwrócenia kodu klawiaturowego jak gdyby użytkownik nacisnął klawisz na klawiaturze.

Używając tej techniki dostarczamy największej kompatybilności (z istniejącym oprogramowaniem) przy zwracaniu kodu klawiaturowego do aplikacji. Niestety, ta sztuczka działa na maszynach mających kontrolery , które są kompatybilne z PS/2; nie jest to większość maszyn. Jednakże jeśli napiszemy kod dla PS/2 lub kompatybilnych, jest to najlepszy sposób.

Kontroler klawiatury na PC/AT i większości innych kompatybilnych maszynach PC nie wspiera polecenia 0D2h. Niemniej jednak jest to podstępny sposób wymuszenia na kontrolerze klawiatury przekazanie kodu klawiaturowego, jeśli złamiemy kilka zasad. Sztuczka ta może nie działać na wszystkich maszynach (jest wiele maszyn na których ta sztuczka jest znana jako błąd), ale jest dostępna na dużej liczbie kompatybilnych maszyn PC.

Sztuczka jest prosta . Chociaż kontroler klawiatury nie ma polecenia do zwracania bajtu, wysyłamy go, dostarcza polecenia do zwrotu bajtu polecenia kontrolera klawiatury (KCCB). Dostarcza również innego polecenia do zapisu wartości do KCCB. Przez zapisanie wartości do KCCB a potem wydanie polecenia odczyty KCCB możemy oszukać system wprowadzając kod programowalny użytkownika. Niestety KCCB zawiera pewne zarezerwowane, niezdefiniowane bity, które mają różne znaczenie na różnego rodzaju chipach mikrokontrolera klawiatury. Jest to główny powód, że technika ta nie działa na wszystkich maszynach. Poniższy kod asemblerowy demonstruje jak używać tych metod dla PS/2 i kontrolera klawiatury PC:

```
                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list

cseg            segment para  public 'code'
                assume  ds:nothing

;*****************************************************************************
;
; PutInATBuffer-
;
; Poniższy kod wkłada kod klawiaturowy do chipa mikrokontrolera klawiatury klasy AT i zapytuje go czy wyśle
; kod klawiaturowy z powrotem do nas (poprzez sprzętowy port)
;
; Kontroler klawiatury AT:
;
; Port danych jest pod adresem I/O 60h
; Port stanu jest pod adresem I/O 64h  (tylko odczyt)
; Port polecenia jest pod adresem I/O 64h (tylko zapis)
;
; Kontroler odpowiada poniższymi wartościami wysyłanymi do portu polecenia:
;
; 20h – Odczyt bajtu polecenia kontrolera klawiatury (KCCB) i wysłanie danych do portu danych (adres I/O 64h)
;
; 60h – zapis do KCCB. Kolejny bajt zapisywany do adresu I/O 60h jest umieszczany w KCCB. Bity w KCCB
; są definiowane jak następuje :
;
;       bit 7-  Zarezerwowane, powinno być zero
;       bit 6-  Tryb komputera przemysłowego
;       bit 5-  Tryb komputera przemysłowego
;       bit 4-  Blokowanie klawiatury
;       bit 3-  Zakaz przykrycia
;       bit 2-  System flag
;       bit 1-  Zarezerwowane, powinno być zero
;       bit 0-  Odblokowanie bufora wyjściowego pełnego przerwań
;
;       AAh -   Autotest
;       ABh  -  Test interfejsu
;       ACh   - Zrzut diagnostyczny
;       ADh  -  Zablokowanie klawiatury
;       AEh   - Odblokowanie klawiatury
;       C0h    - Odczyt portu wejściowego Kontrolera Klawiatury
;       D0h   - Odczyt portu wyjściowego Kontrolera Klawiatury
;       D1h   - Zapis do portu wyjściowego Kontrolera klawiatury
;       E0h   - Odczyt testów wejściowych
;       F0h – FFh -  Impuls portu wyjściowego
;
; Port wyjściowy kontrolera klawiatury jest zdefiniowany jak następuje:
;
;       bit 7-  Dane klawiatury (wyjście)
;       bit 6-  Zegar klawiatury (wyjście)
;       bit 5-  Bufor wejściowy pusty
;       bit 4-  Bufor wyjściowy pełny
;       bit 3-  niezdefiniowany
;       bit 2-  niezdefiniowany
;       bit 1-  Gate A20
;       bit 0-  Reset systemu (0 = reset)
;
```

```
; Port wejściowy kontrolera klawiatury jest zdefiniowany jak następuje:
;
;       bit 7-    Zakazane przełączanie klawiatury (0 = zabronione)
;       bit 6-    Przełączanie monitora ( 0 = kolor, 1 = mono)
;       bit 5-    Zworka
;       bit 4-    RAM płyty głównej (0= zablokowany 256k RAM na płycie głównej)
;       bity 0-3  Niezdefiniowane
;
; Port stanu kontrolera klawiatury (64h) jest zdefiniowany jak następuje:
;
;       bit 1 – Ustawiony jeśli dana wejściowa (60h) nie jest dostępna
;       bit 2-  Ustawiony jeśli port wyjściowy (60h) nie może uzyskać dostępu do danej

PutInATBuffer   proc    near
                assume  ds.:nothing
                pushf
                push    ax
                push    bx
                push    cx
                push    dx

                mov     dl, al                  ;zachowanie znaku na wyjście

;Czekamy dopóki kontroler klawiatury nie  będzie zawierał danej przed kontynuowaniem

                xor     cx, cx
WaitWhlFull:    in      al,  64h
                test    al, 1
                loopnz  WaitWhlFull

;Najpierw maskujemy chip kontrolera przerwań (8259) aby powiedzieć mu o ignorowaniu
; przerwań pochodzących  z klawiatury. Jednakże włączając przerwania właściwie przetwarzamy
l przerwania z innych źródeł (jest to ważne jeśli będziemy wysyłać fałszywe EOI do kontrolera przerwań
; wewnątrz podprogramu BIOS INT 9

                cli
                in      al., 21h        ;pobranie bieżącej maski
                push    ax              ; zachowanie maski przerwań
                or      al., 2          ; maska przerwania klawiatury
                out     21h, al.

; Przekazujemy żądany kod klawiaturowy do kontrolera klawiatury. Wywołujemy ten bajt nowym
; poleceniem kontrolera klawiatury (wyłączyliśmy klawiaturę, więc to nie wpływa na nic)
;
; Poniższy kod mówi kontrolerowi klawiatury aby pobrał kolejny bajt i wysłał do niego i użył tego
; bajtu jako KCCB:

                call    WaitToXmit
                mov     al., 60h        ;zapis nowego polecenia KCCB
                out     64h, al.

;Wysyłamy kod klawiaturowy jako nowy KCCB:

                call    WaitToXmit
                mov     al., dl
                out     60h, al

; Poniższy kod instruuje system o przekazaniu KCCB (tj. kodu klawiaturowego) do systemu:

                call    WaitToXmit
```

```
                mov     al., 20h                ;polecenie “Wysłania KCCB”
                out     64h, al.

                xor     cx, cx
Wait4OutFull:   in      al, 64h
                test    al, 1
                loopz   Wait4OutFull

;Okay, wysyłamy 45h  z powrotem jako nowy KCCB pozwalając normalnej klawiaturze pracować
; poprawnie
                call    WaitToXmit
                mov     al., 60h
                out     64h, al.

                call    WaitToXmit
                mov     al, 45h
                out     60h, al

;Okay wykonujemy podprogram INT 9 więc BIOS (lub kto inny0 może odczytać klawisz jaki właśnie
; upchnęliśmy w kontrolerze klawiatury. Ponieważ zamaskowaliśmy INT 9 w kontrolerze przerwań, nie będzie
; żadnych przerwań pochodzących z klawisza upchniętych w buforze.


DoInt9:         in      al. 60h                 ;zachowanie przerwań w kodzie
                int     9                       ;symulowanie sprzętowych przerwań klawiatury

; Odblokowujemy klawiaturę

                call    WaitToXmit
                mov     al., 0aeh
                out     64h, al.

; Okay, przywracamy maskę przerwania dla klawiatury w 8259a

                pop     ax
                out     21h, al.

                pop     dx
                pop     cx
                pop     bx
                pop     ax
                popf
                ret
PutInATBuffer   endp

; WaitToXmit- czekamy dopóki jest OK, wysyłania bajtu polecenia do portu kontrolera klawiatury

WaitToXmit      proc    near
                push    cx
                push    ax
                xor     cx, cx
TstCmdPortLp:   in      al, 64h
                test    al, 2                   ; sprawdzamy pełny bufor wejściowy flag
                loopnz  TstCmdPortLp
                pop     ax
                pop     cx
                ret
WaitToXmit      endp

;*****************************************************************************************
```

```
;
; PutInPS2Buffer – Podobnie jak PutInATBuffer, używa chipu mikrokontrolera klawiatury do zwracania kodu
; klawisza. Jednakże, kompatybilne kontrolery PS/2 mają aktualne polecenie zwrotu kodu klawisza
PutInPS2Buffer  proc    near
                pushf
                push    ax
                push    bx
                push    cx
                push    dx

                mov     dl, al

;czekamy dopóki kontroler klawiatury nie będzie zawierał danej

                xor     cx, cx
WaitWhlFull     in      al, 64h
                test    al, 1
                loopnz  WaitWhlFull

; Poniższy kod mówi kontrolerowi klawiatury aby pobrał kolejny bajt do wysłania i zwrócił go jako kod
; klawiaturowy

                call    WaitToXmit
                mov     al., 0d2h               ; zwracanie polecenia kodu klawiaturowego
                out     64h, al.

; Wysyłanie kodu klawiaturowego

                call    WaitToXmit
                mov     al., dl
                out     60h, al.

                pop     dx
                pop     cx
                pop     bx
                pop     ax
                popf
                ret
PutInPS2Buffer  endp

;Program główny -  Symuluje uderzenia klawisza demonstrujący powyższy kod

Main            proc

                mov     ax, cseg
                mov     ds, ax

                print
                byte    “Simulating keystrokes via Trace Flag”, cr, lf
                byte    “This program places ‘DIR’ in the keyboard buffer”
                byte    cr, lf, 0

                mov     al, 20h                 ; dolny kod “D”
                call    PutInATBuffer
                mov     al., 0a0h               ;górny kod “D”
                call    PutInATBuffer

                mov     al., 17h                ;dolny kod “I”
                call    PutInATBuffer
                mov     al., 97h                ;górny kod “I”
```

```
                call    PutInATBuffer

                mov     al., 13h                ;dolny kod "R"
                call    PutInATBuffer
                mov     al., 93h                ;górny kod "R"
                call    PutInATBuffer

                mov     al., 1Ch                ;dolny kod Enter
                call    PutInATBuffer
                mov     al., 9Ch                ;górny kod  Enter
                call    PutInATBuffer

                ExitPgm
Main            endp

cseg            ends

sseg            segment para stack 'stack'
stk             byte    1024  dup ("stack")
sseg            ends

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       db      16  dup (?)
zzzzzzseg       ends
                end     Main
```

20.8 PODSUMOWANIE

Rozdział ten może wydawać się nadmiernie długi jak na tak przyziemny I/O klawiatury. W końcu Biblioteka Standardowa dostarcza tylko jeden, prosty podprogram dla klawiatury , getc. Jednakże klawiatura PC jest bestią złożoną, mającą nie mniej niż dwa wyspecjalizowane mikroprocesory nim sterujące. Mikroprocesory te akceptują polecenia z PC i wysyłają polecenia i dane do PC. Jeśli chcemy napisać jakiś skomplikowany kod obsługi klawiatury, musimy dobrze rozumieć klawiaturę sprzętu bazowego.

Rozdział ten zaczyna się opisem działania systemu kiedy użytkownik naciśnie klawisz. Okazuje się , że system przekazuje dwa kody klawiaturowe za każdym razem kiedy naciskamy klawisz – jeden kod klawiaturowy kiedy naciskamy klawisz i jeden kiedy zwalniamy klawisz. Są one nazwane dolnym i górnym kodem, odpowiednio. Kody klawiaturowe przekazywane do systemu mają trochę powiązań ze standardowym zbiorem znaków ASCII. Zamiast tego, klawiatura używa swojego własnego zbioru a podprogram obsługi przerwania klawiaturowego tłumaczy te kody klawiaturowe na ich właściwe kody ASCII. Niektóre klawisze nie mają kodów ASCII, dla tych klawiszy system przekazuje rozszerzony kod klawisza do aplikacji żądającej wejścia z klawiatury. Podczas tłumaczenia kodu klawiaturowego na kod ASCII, ISR klawiaturowy stosuje pewne flagi BIOS'a , które śledzą pozycję klawiszy modyfikujących. Do klawiszy tych zalicza się klawisze shift, ctrl, alt, capslock i numlock. Klawisze te są znane jako modyfikujące ponieważ modyfikują normalny kod tworzony przez klawisze na klawiaturze . ISR klawiaturowy upycha nadchodzące znaki w systemowym buforze roboczym i aktualizuje inne zmienne BIOS w segmencie 40h. Aplikacja lub inny system usług może mieć dostęp do tej danej przygotowanej przez podprogram obsługi przerwań klawiaturowych.

*"Podstawy klawiatury"

Interfejs PC klawiatury używa dwóch oddzielnych chipów mikrokontrolerów. Chipy te dostarczają użytkownikowi rejestrów programowych i bardzo elastycznego zbioru poleceń. Jeśli chcemy oprogramować klawiaturę poza prostym odczytem naciśnięć klawiszy (np. manipulowanie LED'ami na klawiaturze), będziemy musieli bliżej się zapoznać z tymi rejestrami i zbiorem poleceń tych mikrokontrolerów

*"Interfejs sprzętowy klawiatury"

Oba, DOS i BIOS dostarczają umiejętności odczytu klawisza z systemowego bufora roboczego. Jaka zwykle funkcje BIOS'a dostarczają elastyczności pod względem osiągania sprzętu. Co więcej, podprogram BIOS int 16h pozwala nam sprawdzić stan klawisza shift, wkłada kody klawiaturowe/ ASCII do bufora roboczego, modyfikując częstotliwość autopowtarzania i więcej. Mając tą elastyczność, trudno jest zrozumieć

dlaczego ktoś chciałby komunikować się bezpośrednio ze sprzętem klawiaturowym, zwłaszcza zważywszy na problemy kompatybilności, które wydają się plagą takich projektów. Aby nauczyć się właściwego sposobu odczytu znaku z klawiatury zajrzyj:

*"Interfejs DOS klawiatury"
*"Interfejs BIOS klawiatury"

Chociaż bezpośredni dostęp do sprzętu klawiatury jest złym pomysłem dla większości, jest mała klasa programów, jak poprawiona klawiatura i programy pop-up, które rzeczywiście muszą uzyskać bezpośredni dostęp do sprzętu klawiaturowego. Programy te muszą dostarczyć podprogramu obsługi przerwania dla przerwania (klawiatury) int 9

*"Podprogram obsługi przerwań klawiaturowych"
*"Aktualizacja podprogramu obsługi przerwania INT 9"

Program klawiatury makro (poprawiona klawiatura) jest doskonałym przykładem programu, który może musieć komunikować się bezpośrednio ze sprzętem klawiatury. Jeden problem z takimi programami polega na tym, że muszą przekazywać znaki do podstawowej aplikacji. Znając naturę aplikacji obecnych w świecie, może to być trudnym zadaniem, jeśli chcemy mieć kompatybilność z dużą liczbą aplikacji PC. Te problemy i pewne rozwiązania pojawią się w:

*"Symulowanie uderzeń w klawisze"
*"Wstawianie znaków do bufora roboczego"
*"Używanie flagi śledzenia 80x86 do symulowania instrukcji IN AL., 60H"
*"Używanie mikrokontrolera 8042 do symulowania uderzeń w klawisze"