ROZDZIAŁ DWUDZIESTY CZWARTY: **ZŁĄCZE GIER PC**

Ktoś kto zajrzy głębiej wewnątrz kilku popularnych gier PC odkryje, że wielu programistów nie w pełni rozumie jedno z najmniej złożonych urządzeń dołączonych do dzisiejszych PC – analogowego złącza gier. Urządzenie to pozwala użytkownikowi połączyć do czterech potencjometrów rezystancyjnych i czterech przełączników cyfrowych połączonych z PC. Na projekt złącza gier PC wpłynęły oczywiście możliwości wejścia analogowego komputera Applle II, najpopularniejszego komputera dostępnego w czasie prac nad PC. Chociaż IBM dostarczył da razy więcej wejść analogowych niż Apple II, decyzja wsparcia tylko czterech przełączników i czterech potencjometrów (lub „pots") wydaje się ograniczać dzisiejszych projektantów gier – w ten sam sposób co decyzja IBM o wsparciu 256 K RAM wydaje się dzisiaj ograniczeniem. Niemniej jednak, projektanci gier dają sobie radę tworząc rzeczywiście cudowne produkty, nawet żyjąc z ograniczeniami IBM z 1981 roku.

Projekt IBM'owskiego wejścia analogowego, podobnie jak Apple, zaprojektowano za psie pieniądze. Dokładnością i wydajnością nie przejmowano się wcale. Faktycznie możemy kupić elektroniczne części do zbudowania swojej wersji złącza gier, detalicznie, po około trzy dolary. Istotnie możemy dzisiaj zakupić kartę złącza gier z różnymi kupieckimi rabatami za około osiem dolarów. Niestety, mało kosztowny, IBM'owski projekt z 1981 roku stwarza problemy z wydajnością dla szybkich maszyn i wysoko wydajnych gier w 1990 roku. Jednakże nie ma co rozpaczać na d rozlanym mlekiem – zostaniemy przy projekcie oryginalnego złącza gier. Poniższa sekcja opisuje dokładnie jak go wykorzystać.

24.1 TYPOWE URZĄDZENIA DO GIER

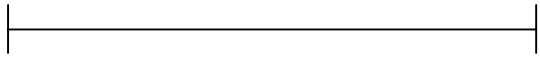
Złącze gier jest niczym więcej jak sprzęgiem komputerowym dla różnych urządzeń dla gier. Typowa karta złącza gier zawiera złącze DB15 do którego możemy podpiąć urządzenie zewnętrzne Do typowych urządzeń jakie możemy podłączyć do złącza gier zaliczamy paddle, joystick, flight yoke, cyfrowy joystick, rudder pedal, symulator RC i steering wheels. Bez wątpienia jest to krótka lista typowych urządzeń jakie możemy podłączyć do złącza gier. Wiele z tych urządzeń jest o wiele droższych niż sama karta złącza gier. Rzeczywiście, wysokiej jakości konsole symulatorów lotu dla złącza gier kosztują kilka set dolarów.

Joystick cyfrowy jest prawdopodobnie najmniej złożonym urządzeniem jakie możemy podłączyć do portu gier PC. Urządzenie to składa się z czterech przełączników i drążka. Przesuwanie drążka w przód lewo, prawo lub w tył, zamyka jeden z tych przełączników Karta złącza gier dostarcza czterech wejść przełączników, wiec możemy wyczuć kierunek (wliczając w to pozostałe pozycji) użytkownika naciskającego joystick. Większość cyfrowych joysticków pozwala również na wyczucie pozycji pośrodku poprzez zwarcie dwóch kontaktów na raz, na przykład poprzez ułożenie drążka pod kątem 45 stopni pomiędzy pozycją w „przód" a „w prawo". Aplikacje mogą to odczuć i wybrać odpowiednią akcję. Pierwszą korzyścią tych urządzeń jest to ,że są bardzo tanie w wytworzeniu (dlatego pierwsze joysticki znajdowały się w większości domowych maszyn). Jednakże, producenci zwiększając produkcje joysticków analogowych ceny spadły do punktu w którym joysticki cyfrowe nie zdołały zaoferować pokaźniej różnicy cen. Wiec dzisiaj rzadko możemy spotkać takie urządzenia w rękach użytkowników.

Paddle jest innym urządzeniem , którego używanie podupadło przez lata. **Paddle** jest pojedynczym potencjometrem z pojedynczym pokrętłem (i zazwyczaj z jednym przyciskiem) . Apple dostarczał pary paddle z każdym sprzedanym Apple II. W wyniku tego, gry używające paddle były całkiem popularne kiedy IBM wypuścił PC w 1981 roku. Istotnie klika firm [produkowało paddle dla PC, kiedy został wypuszczony po raz pierwszy. Jednakże, znowu koszt wytworzenia joysticków analogowych spadł do poziomu z którym paddle nie mogło konkurować. Chociaż paddle są odpowiednimi urządzeniami wejściowymi dla wielu gier, joysticki mogą robić wszystko to co paddle i wiele innych. W tej sytuacji stosowanie paddle szybko zamiera. Jest jedna rzecz,

jaką możemy zrobić paddle , a której nie można zrobić joystickiem – możemy umieścić cztery z nich w systemie i stworzyć czterech graczy. Jednak to (oczywiście0 nie jest ważne dal większości projektantów gier, którzy generalnie projektują gry tylko dla jednego gracza.

Paddle lub zbiór rudder pedals
generalnie dostarczają pojedynczej liczby
z zakresu od zera do jakiejś wartości maksymalnej
zależnej od systemu

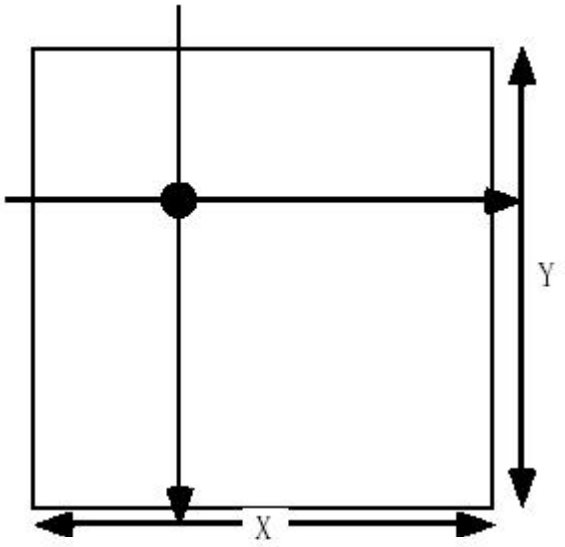


0 Odczyt maksimum

Urządzenie wejściowe paddle lub rudder pedal

Rudder pedals są niczym więcej niż specjalnie zaprojektowanym paddle'em, zaprojektowanym tak aby można było aktywować je stopami. Wiele gier symulatorów lotu wykorzystuje to urządzenie wejściowe dostarczając wiele rzeczywistych przeżyć. Ogólnie będziemy używali rudder pedals jako dodatek do joysticka.

Joystick zawiera dwa potencjometry połączone z drążkiem. Przesunięcie joysticka analogowego wzdłuż osi x uruchamia jeden z potencjometrów, przesuwając joystick wzdłuż osi y uruchamia inny potencjometr. Poprzez odczyt obu potencjometrów możemy mniej więcej określić absolutną pozycję potencjometrów wewnątrz ich zakresu pracy.



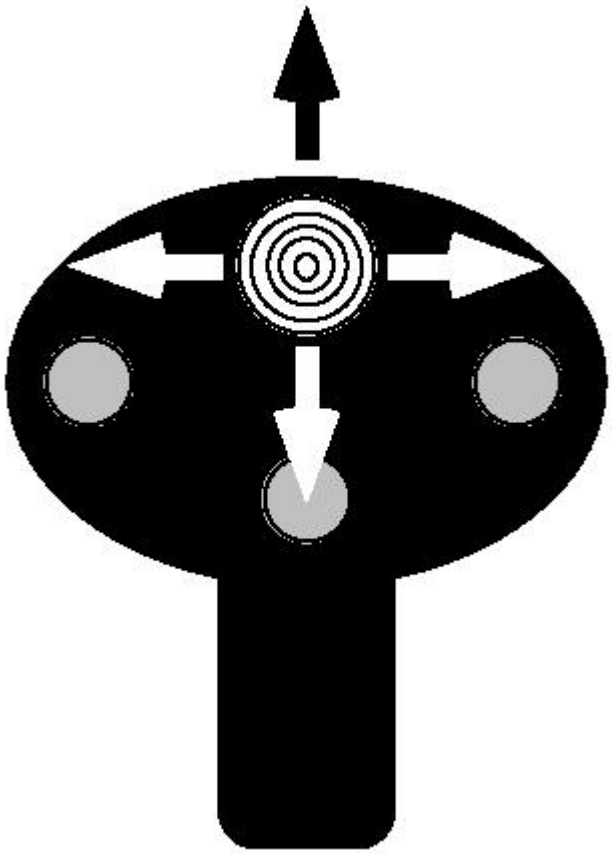
URZĄDZENIE WEJŚCIOWE JOYSTICK

Joystick używa dwóch niezależnych potencjometrów dostarczających wartości wejściowych (X,Y). Poziome przesunięcie joysticka wpływa na oś x potencjometru nie zależnie od osi y potencjometru. Podobnie pionowe przesunięcie wpływa na oś y niezależnie od osi x potencjometru. Przez odczytanie obu potencjometrów możemy określić pozycję joysticka w systemie współrzędnych (X,Y).

Symulator RC jest niczym więcej niż pudełkiem zawierającym dwa joysticki . Urządzenia **yoke** i **steering wheel** są zazwyczaj takimi samymi urządzeniami sprzedawanymi specjalnie do symulatorów lotu lub gier samochodowych. **Steering wheel** jest podłączony do potencjometru, który odpowiada osi x joysticka. Cofnięcie (lub popchnięcie) na kierownicy aktywuje drugi potencjometr odpowiadający osi y joysticka.

Niektóre urządzenia joystickowe, ogólnie znane jako **flight sticks** zawiera trzy potencjometry . Dwa potencjometry są połączone w sposób standardowego joysticka, trzeci jest podłączone do gałki, której wiele gier używa dla sterowania przepustowością . Inne joysticki, takie jak Thrustnmaster™ lub CH Products'FlightStick Pro, zawierają dodatkowe przełączniki zawierające specjalne „**cooley switch"**, które dostarczają dodatkowych wejść do gry. **Cooley switch** jest, w gruncie rzeczy, cyfrowym potencjometrem zamontowanym na szczycie joysticka. Użytkownicy mogą wybrać jedną z czterech pozycji **cooley switch'a**

używając kciuków. Większość programów symulatorów lotu zgodnych z takimi urządzeniami używa **cooley switcha** do wyboru różnych widoków z samolotu.

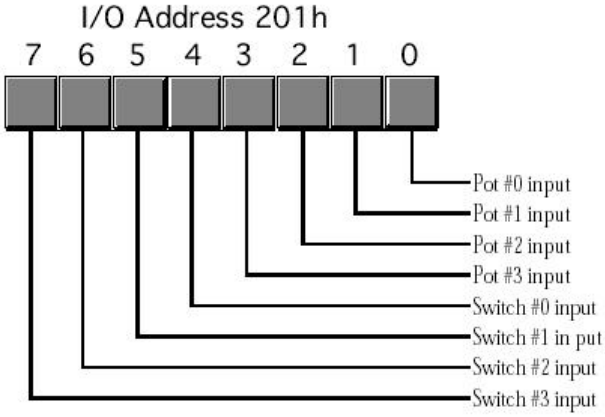


COOLEY SWITCH

Cooley switch (pokazany powyżej na urządzeniu podobnym do CH Products' FlightStick Pro) uruchamia kciukiem cyfrowy joystick. Możemy przesunąć przełącznik w górę, dół, w lewo lub prawo, uruchamiając pojedyncze przełączniki wewnątrz urządzenia.

24.2 SPRZĘTOWE ZŁĄCZE GIER

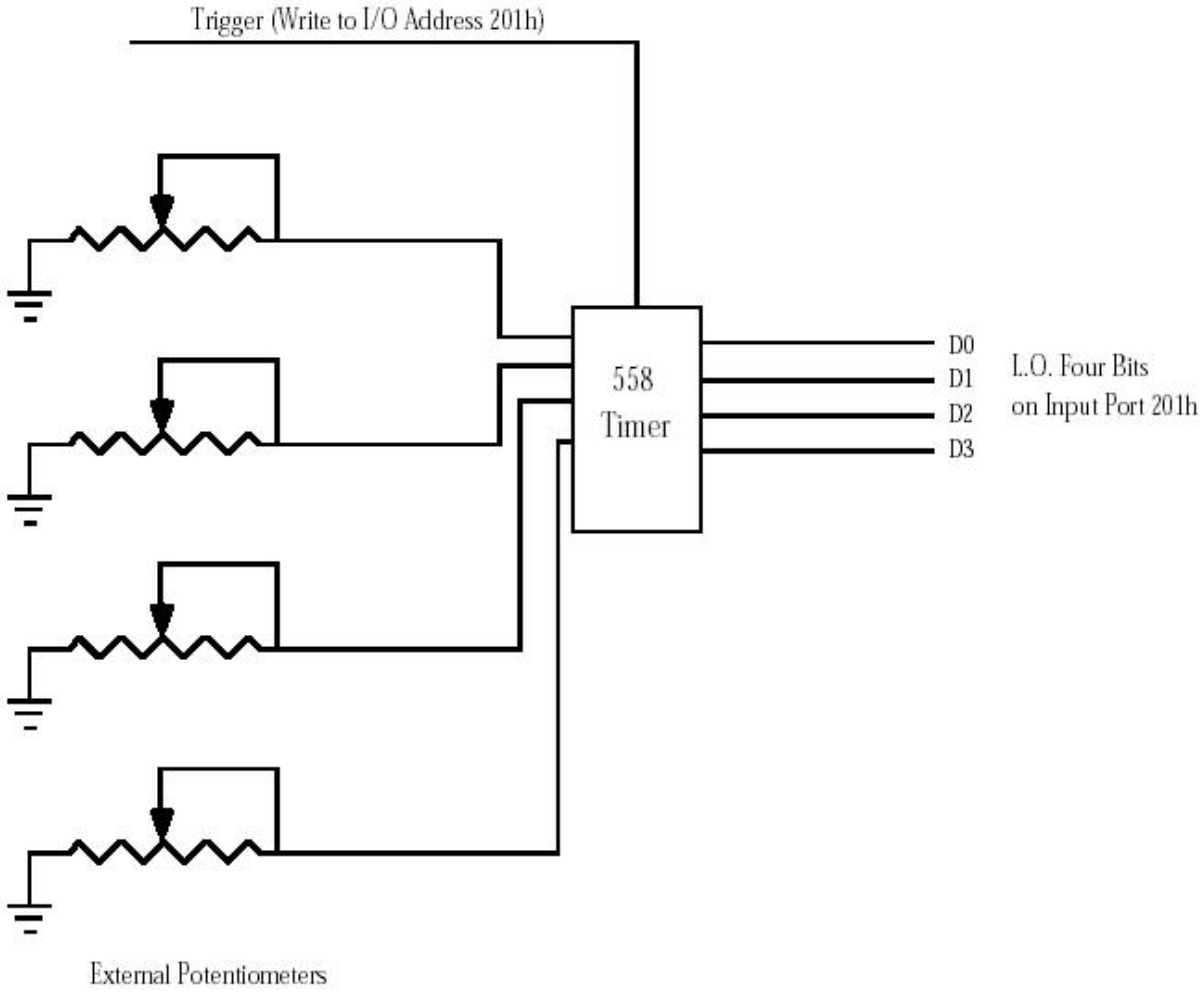
Sprzętowe złącze do gier jest proste. Jest to pojedynczy port wejściowy i pojedynczy port wyjściowy. Rozkład bitów portu wejściowego jest następujący



GAME ADAPTER INPUT PORT

Cztery przełącznik nadchodzą do czterech bardziej znaczących bitów portu I/O 201h. Jeśli użytkownik aktualnie nacisnął przycisk, odpowiednie pozycje bitów będą zawierały zero. Jeśli przycisk jest zwolniony, odpowiednie bity będą zawierały jeden

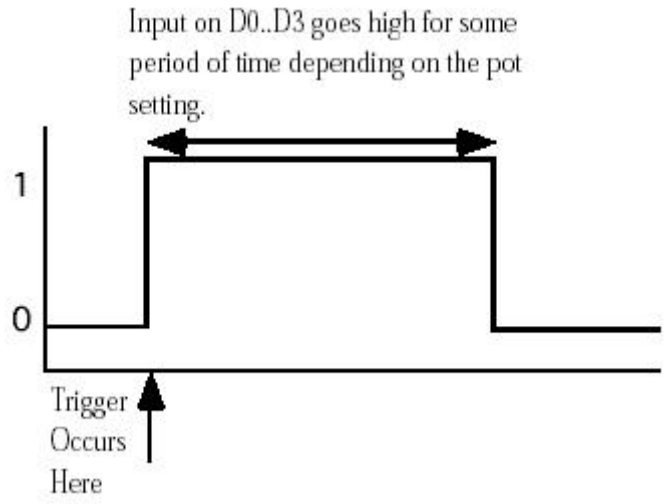
Potencjometr wejściowy może wydawać się dziwny na pierwszy rzut oka. W końcu, jak można przedstawić jedną z dużych liczb potencjalnej pozycji potencjometru (powiedzmy, 256) pojedynczym bitem? Oczywiście nie możemy. Jednakże, bit wejściowy w tym porcie nie zwraca żadnego typu wartości numerycznej określającej pozycję potencjometru. Zamiast tego, każdy z czterech bitów potencjometru jest połączony z wejściem wrażliwego na oporność czterowyjściowego chipu zegarowego 558. Kiedy wyzwalamy chip zegarowy, tworzy on impuls wyjściowy o czasie trwania proporcjonalnym do oporności wejścia do timera. Wyjście tego chipu zegarowego jest traktowane jako bit wejściowy danego portu .Schemat tego układu to



SCHEMAT JOYSTICKA

Normalnie bity wejściowe joysticka zawierają zero. Kiedy wyzwalamy chip zegarowy, linie wejściowe potencjometru idą wyżej dla tego samego okresu czasu określonego przez bieżącą oporność potencjometru. Poprzez pomiar jak długo ten bit pozostaje ustawiony, możemy uzyskać wstępne oszacowanie oporności. Aby wyzwolić potencjometr po prostu zapisujemy wartość do portu I/O 201h. Rzeczywista wartość jaką wpiszemy jest nieistotna

Poniższy wykres pokazuje sygnał różni się na każdym z bitów wejściowych potencjometru



ANALOGOWY SYGNAŁ WEJŚCIOWY CZĘSTOTLIWOŚCI ZEGARA

Pozostaje jedynie pytanie „jak określimy długość impulsu?”. Poniższa krótka pętla demonstruje jeden sposób określenia szerokości tego impulsu:

```
        mov     cx, -1          ;Mamy zamiar liczyć wstecz
        mov     dx, 201h        ;wskazujemy port joysticka
        out     dx, al.         ;wyzwalamy chip zegarowy
CntLp:  in      al., dx         ;odczyt portu joysticka
        test    al 1            ;sprawdzenie wejścia # 0 potencjometru
        loopne  CntLp           ;powtarzanie dopóki wysoko
        neg     cx              ;konwertujemy CX do wartości dodatniej
```

Kiedy ta pętla kończy wykonywanie, rejestr cx będzie zawierał liczbę przejść uczynionych przez tą pętle podczas gdy sygnał wyjściowy timera był logiczną jedynką. Duża wartość w cx, , dłuższy impuls i dlatego większa oporność potencjometru # 0.

Jest kilka ważnych problemów z tym kodem. Przede wszystkim, kod ten oczywiście będzie tworzył różne wyniki na różnych maszynach działających przy różnych częstotliwościach cyklu zegara. Na przykład, system Pentium 150 MHz będzie wykonywał ten kod dużo szybciej niż system 8088 5 MHz. Drugi problem jest taki, że joysticki i inne karty adaptera gier tworzą radykalnie różne wyniki częstotliwości zegara. Nawet w tym samym systemie z taką samą kartą adaptera i joystickiem, nie musimy zawsze uzyskiwać spójnych odczytów w różnych dniach. Okazuje się ,że 558 jest w pewnym sensie czuły na temperaturę i będzie tworzył odrobinę różne odczyty przy zmianach temperatury.

Niestety, nie ma sposobu stworzenia takiej pętli jak powyższa aby zwracała stały odczyt dla szerokiego wyboru maszyn, potencjometrów i kart adapterów gier. Dlatego też musimy napisać naszą aplikację tak aby była niewrażliwa na szerokie zmiany w wartościach wejściowych z wejść analogowych. Na szczęście, js to bardzo łatwe do zrobienia, ale więcej o tym później.

24.3 ZASTOSOWANIE FUNKCJI BIOS GIER I/O

BIOS dostarcza dwóch funkcji do odczytu wejść złącza gier. Obie są podfunkcjami programu obsługi int 15h.

Do odczytu przełączników, ładujemy do ah 84h a do dx zero, potem wykonujemy instrukcję int 15h. Przy zwrocie, al będzie zawierał odczytane cztery bardziej znaczące bity przełącznika (zobacz wykres w poprzedniej sekcji). Funkcja ta jest w przybliżeniu odpowiednikiem bezpośredniego odczytu portu 201h.

Dla odczytu wejść analogowych, ładujemy do ah 84h a do dx jedynką potem wykonujemy instrukcję int 15h Przy zwrocie AX,BX,CX i DX będą zawierały wartości, odpowiednio, dla potencjometrów zero, jeden, dwa i trzy. W praktyce, funkcja to powinna zwrócić wartości z zakresu 0 –400h, chociaż nie możemy liczyć na to z powodów opisanych w poprzedniej sekcji.

Bardzo niewiele programów używa wsparcia joysticka BIOS. Łatwiej odczytać przełączniki bezpośrednio a odczyt potencjometrów nie jest tą pracą, która wywołuje podprogramy BIOS. Kod BIOS jest bardzo wolny. Większość BIOS’ów odczytuje sekwencyjnie cztery potencjometry, wykonując to do czterech

razy dłużej niż program, który odczytuje wszystkie cztery potencjometry równocześnie (zobacz następną sekcję) Ponieważ odczyt potencjometrów może trwać kilkaset mikrosekund do kilku milisekund, większość programistów pisze wysokowydajne gry nie używając funkcji BIOS, piszą oni swoje własne wysoko wydajne podprogramy.

To prawdziwy wstyd. Poprzez napisanie określonych sterowników do oryginalnych gier PC złącza gier, ci programiści zmuszają użytkownika do kupowania i użytkowania standardowych kart złącza gier i urządzeń gier. Gdyby byłyby to gry wywołujące funkcje BIOS, trzecia część programistów może stworzyć różne i unikalne sterowniki gier a potem po prostu wesprzeć sterowniki, które zamieniają podprogram int 15h i dostarczyć takiego samego interfejsu programistycznego. Na przykład, Genovation stworzył urządzenie, któ®e pozwala nam podłączyć joystick do portu równoległego PC. Colorado Spectrum stworzyło podobne urządzenie, któ®e pozwala podpiąć joystick do portu szeregowego. Oba urządzenia pozwolą nam na zastosowanie joysticka na maszynie, która nie ma (i być może nie może mieć) zainstalowanego złącza gier. Jednakże, gry, uzyskujące bezpośredni dostęp do joysticka sprzętowego, nie będą kompatybilne z takimi urządzeniami. Jednak projektując gry w oparciu o funkcję int 15h, oprogramowanie byłoby kompatybilne ponieważ zarówno Colorado Spectrum i Genovation wspierają TSR'y int 15h dla przekierowania wywołania joysticka do zastosowania w tych urządzeniach.

Aby pomóc w przezwyciężeniu niechęci projektantów gier do stosowania int 15h, tekst ten będzie prezentował wysoko wydajne wersje kodu joysticka BIOS trochę później w tym rozdziale. Programiści, którzy adoptują Standard Game Device Interface stworzą programy, które będą kompatybilne z innymi urządzeniami, które wspierają standard SGDI. Po więcej szczegółów, zobacz do „Standardowy Interfejs Urządzenia Gier (SGDI)".

24.4 PISANIE WŁASNEGO PODPROGRAMU I/O GIER

Rozważmy ponownie kod, który zwraca pewną wartość dla danego ustawienia:

```
        mov     cx,  -1                 ; mamy zamiar liczyć wstecz
        mov     dx, 201h                ;wskazanie portu joysticka
        out     dx, al.                 ;wyzwolenie chipu zegarowego
CntLp:  in      al., dx                 ;odczyt portu joysticka
        test    al, 1                   ;sprawdzenie wejścia potencjometru # 0
        loopne  CntLp                   ;powtarzanie dopóki wysoko
        neg     cx                      ;konwersja CX do wartości dodatniej
```

Jak wspomniano wcześniej, dużym problemem z tym kodem jest to ,że mamy zamiar pobrać gwałtownie wartości o różnych zakresach z różnych kart złączy gier, urządzeń wejściowych i systemów komputerowych. Oczywiście nie może zawsze liczyć na powyższy kod tworząc wartości w zakresie 0 ...180h pod takimi warunkami. Nasze oprogramowanie będzie musiało dynamicznie modyfikować wartości używane w zależności od parametrów systemu.

Prawdopodobnie będziemy grac w gry na PC, gdzie oprogramowanie prosić będzie o kalibrację joysticka przed użyciem. Kalibracja generalnie składa się z przesunięcia joysticka do jednego z rogów (na przykład lwy górny róg), naciśnięcia przycisku lub klawisza a potem przesunięcie do przeciwległego rogu (np. dół-prawo) i ponownego naciśnięcia przycisku. Pewne systemy chcą nawet przesunięcia joysticka do pozycji centralnej i naciśnięcia przycisku.

Oprogramowanie, które to robi odczytuje minimalną, maksymalną i centralną wartość z joysticka .Mając dana przynajmniej minimalną i maksymalną wartość, możemy łatwo wyskalować odczyt do zakresu jaki chcemy. Poprzez odczyt wartości centralnej, możemy uzyskać odrobinę lepsze wyniki, zwłaszcza na rzeczywiście niedrogich (tanich) joystickach. Ten proces skalowania odczytów do pewnego zakresu jest znany jako normalizacja. Poprzez odczyt wartości minimalnej i maksymalnej od użytkownika i potem normalizacji każdego odczytu, możemy napisać program zakładając, że wartość zawsze się będą znajdować wewnątrz pewnego zakresu, na przykład 0...255.Normalizacja odczytu jest bardzo łatwa, po prostu używamy następującej formuły:

$$\frac{(\text{BieżącyOdczyt} - \text{OdczytMinimalny})}{(\text{OdczytMaksymalny} - \text{OdczytMinimalny})} \times \text{WartośćNormalna}$$

Wartości OdczytMaksymalny i Odczyt Minimalny są wartościami minimalną i maksymalną odczytanymi od użytkownika na początku naszej aplikacji. BieżącyOdczyt jest to wartość odczytana ze złącza gier. WartośćNormalna jest to górna granica zakresu, do jakiego chcemy znormalizować odczyt (np. 255), dolną granicą jest zawsze zero

Dla uzyskania lepszych wyników , zwłaszcza kiedy używamy joysticka, powinniśmy uzyskać trzy odczyty podczas fazy kalibracji dla każdego potencjometru – wartość minimalną, wartość maksymalną i wartość centralną. Dla normalizacji odczytu, kiedy uzyskaliśmy te trzy wartości, powinniśmy użyć poniższych formuł:]

Jeśli bieżący odczyt jest w zakresie minimalna...centralna , używamy wzoru:

$$\frac{\text{(Bieżąca – Centralna)}}{\text{(Centralna – Minimalna) x2}} \times \text{WartośćNormalna}$$

Jeśli bieżący odczyt jest w zakresie centralna...maksymalna używamy wzoru:

$$\frac{\text{(Bieżąca – Centralna)}}{\text{(Maksymalna – Centralna) x2}} \times \text{WartośćNormalna} + \frac{\text{WartośćNormalna}}{2}$$

Duża liczba gier na rynku spełnia wszystkie rodzaje wymagań próbując wymusić odczyt joysticka w rozsądnym zakresie. Zaskakująco niewiele z nich używa tej prostej, powyższej formuły. Niektórzy projektanci gier mogą sprzeczać się, że powyższe formuły są zbyt złożone i pisane dla wysoko wydajnych gier. To nonsens. Zabiera dwukrotnie więcej czasu oczekiwanie na przekroczenie czasu przez joystick niż obliczenie powyższego równania. Więc używajmy ich i uczyńmy nasze programy łatwiejszymi w pisaniu.

Chociaż normalizacja naszego odczytu potencjometru zabiera trochę czasu, zawsze jest warta zachodu, odczyt wejść analogowych jest zawsze drogą operacją pod względem cykli CPU. Ponieważ układ zegarowy tworzy relatywnie stały czas opóźnienia dla danej oporności, będziemy nawet marnować więcej cykli CPU na szybkich maszynach niż robiąc to na maszynach wolnych (chociaż odczyt potencjometru zabiera mniej więcej taką samą ilość czasu rzeczywistego na każdej maszynie). Jednym z pewnych sposobów uniknięcia zmarnowania dużej liczby czasu, jest odczyt liku potencjometrów w tym samym czasie; na przykład kiedy odczytujemy potencjometr i jeden uzyskując odczyt joysticka, najpierw odczytujemy potencjometr zero a potem potencjometr jeden. Okazuje się , że możemy łatwo odczytać oba potencjometry równolegle. Robiąc to możemy przyspieszyć odczyt joysticka dwukrotnie. Rozważmy poniższy kod:

```
        mov     cx, 1000h       ;maksymalny czas pętli
        mov     si, 0           ;odkładamy odczyt w SI i DI
        mov     di, si
        mov     ax, si          ;ustawiamy AH na zero
        mov     dx, 201h        ;wskazanie portu joysticka
        out     dx, al.         ;wyzwolenie chipu zegarowego
CntLp:  in      al., dx         ;odczyt portu joysticka
        and     al, 11b         ; usunięcie niepotrzebnych bitów
        jz      Done
        shr     ax, 1           ;wartość potencjometru 0 do flagi przeniesienia
        adc     si, 0           ;zwiększenie wartości pot. 0 jeśli jeszcze aktywny
        add     di, ax          ;zwiększenie wartości pot. 1 jeśli jest aktywny
        loop    CntLp           ;powtarzanie dopóki wysoko
        and     si, 0FFFh       ;jeśli  przekroczono czas, wymuszamy na
        and     di, 0FFFh       ;rejestrach zwierających 1000h do zera
Done:
```

Kod ten odczytuje oba potencjometry zero i jeden w tym samym czasie. Działamy w pętli dopóki każdy potencjometr jest aktywny. Przez cały czas w pętli, kod ten dodaje wartości bitów potencjometrów do oddzielnych rejestrów, które akumulują wynik. Kiedy pętla się kończy, si i di zawierają odczyty dla obu potencjometrów zero i jeden.

Chociaż ta szczególna pętla zawiera więcej instrukcji niż pętla poprzednia, zabiera tyle samo czasu jej wykonanie. Pamiętajmy, że impuls wyjściowy na timerze 558 określa jak długo ten kod się wykonuje, liczba instrukcji w pętli przyczynia się w niewielkim stopniu do czasu wykonania. Jednakże czas tej pętli potrzebny do wykonania jednej iteracji pętli wpływa na decyzję podprogramu odczytu joysticka. Szybsze wykonanie pętli, więcej iteracji pętli będzie uruchomionych podczas takiego samego okresu czasu i będzie lepszy pomiar.

Generalnie chociaż rozdzielczość powyższego kodu jest większa niż dokładność elektronicznych urządzeń gier, więc nie jest to wielkim problemem.

Powyższy kod demonstruje jak odczytać dwa potencjometry. Łatwo jest rozszerzyć ten kod do odczytu trzech lub czterech potencjometrów. Przykład takiego podprogramu pojawi się w sekcji o sterownikach urządzeń SGDI dla standardowej karty rozszerzeń gier.

Inne urządzenia gier, przełączniki, wydają się być prostsze w porównaniu z potencjometrami. Zazwyczaj jednak rzeczy nie są tak łatwe jak wydaje się to na pierwszy rzut oka. Przełączniki mają kilka swoich własnych problemów

Pierwszy to drganie styków klawisza. Przełączniki w typowym joysticku są prawdopodobnie rząd wielkości gorsze niż klawisze na najtańszej klawiaturze. Drgania styków klawiszy jest faktem z jakim musimy się liczyć kiedy odczytujemy przełączniki joysticka. Generalnie, nie powinniśmy odczytywać przełączników joysticka częściej niż raz na 10 ms. Wiele gier odczytuje przełączniki w czasie przerwania zegarowego 55 ms. Na przykład przypuśćmy, że nasze przerwanie zegarowe odczytuje przełączniki i przechowuje wynik w zmiennej pamięciowej. Główna aplikacja, kiedy chcemy odpalić broń, sprawdza tą zmienną. Jeśli jest ustawiona pogram główny czyści zmienną i odpala broń. 55 milisekund później, timer ustawia zmienną przycisku ponownie a program główny odpali ponownie ,kiedy po raz kolejny sprawdzi zmienną. Taki schemat całkowicie wyeliminowałby problemy z drganiem styków klawiszy.

Powyższa technika rozwiązuje inny problem z przełącznikami: śledzenia kiedy przycisk ognia jest w dole .Pamiętajmy, kiedy odczytujemy przełączniki, bity które powróciły mówią nam , że przełącznik jest aktualnie w dole. Nie mówią nam, że przycisk został naciśnięty. Musimy sami to wyśledzić. Jednym z łatwych sposobów wykrycia kiedy użytkownik pierwszy raz nacisnął jest zachowanie poprzedniego odczytu przełącznika i porównanie go z bieżącym odczytem. Jeśli różnią się a bieżący odczyt wskazuje ,że przełącznik jest w dole wtedy jest nowa pozycja dolna.

24.5 STANDARDOWY INTERFEJS URZĄDZENIA GIER (SGDI)

Standardowy Interfejs Urządzenia Gier (SGDI) jest specyfikacją dla usługi 15h, która pozwala nam odczytać przypadkową liczbę potencjometrów i joysticków. Napisanie SGDI zgodnych aplikacji jest łatwe i pomaga uczynić nasze aplikacje zgodne ze sterownikami gier, które dostarczają zgodności z SGDI. Poprzez napisanie aplikacji używających SGDI API możemy zapewnić ,że nasze aplikacje będą działały z przyszłymi sterownikami, które dostarczą poszerzonych możliwości SGDI. Rozumiejąc siłę i rozszerzalność SGDI, musimy przyjrzeć się interfejsowi programowemu aplikacji (API) dla SGDI

24.5.1 INTERFEJS PROGRAMOWY APLIKACJI (API)

Interfejs SGDI rozszerza BIOS joysticka PC o API int 15h. Wykonamy funkcje SGDI poprzez załadowanie rejestru ah 80x86 84h a dx właściwym kodem funkcji SGDI a potem wykonując instrukcję int 15h. Interfejs SGDI po prostu rozszerza funkcjonalność wbudowanych podprogramów BIOS. Zauważmy, że i program, który wywołuje standardowe podprogramy joysticka BIOS będzie działał ze sterownikiem SGDI. Poniższa tablica pokazuje funkcje SGDI:

DH	Wejście	Wyjście	Opis
00	dl =0	al – odczyt przełącznika	Read4Sw. Jest to standardowa podfunkcja BIOS wywołana zerem. Odczytuje stan pierwszych czterech przełączników i zwraca ich wartości górnych czterech bitów rejestru al.
00	dl = 1	ax – pot 0 bx – pot 1 cx – pot 2 dx – pot 3	Read4Pot. Standardowa podfunkcja BIOS wywoływana jedynką. Odczytuje wszystkie cztery potencjometry (jednocześnie) a zwraca niezmodyfikowanych wartości w ax, bx, cx i dx jako specyfikację BIOS
01	dl = pot #	al. = odczyt potencjometru	ReadPot. Funkcja ta odczytuje potencjometr i zawraca znormalizowany odczyt w zakresie 0.255
02	dl =0 al. = maska potencjometru	al. = pot 0 ah = pot 1 dl = pot 2 dh = pot 3	Read4. Podprogram ten odczytuje cztery potencjometry w standardowych kartach rozszerzeń gier podobnie jak powyższa funkcja Read4Pots. Jednakże ten podprogram normalizuje cztery wartości z zakresu 0..255 i zwraca te wartości w al., ah, dl i dh Na wejściu rejestr al. zawiera „maskę potencjometru”, której możemy użyć do wyboru, który z czterech potencjometrów ten podprogram aktualnie odczytuje.
03	dl = pot # al. = minimum bx = maximum		Kalibracja. Funkcja ta kalibruje potencjometry dla tych funkcji, które zwracają znormalizowane wartości. Musimy skalibrować potencjometry przed wywołaniem takiej funkcji potencjometru (ReadPot i Read4) Wartości wejściowe muszą być

	cx = centralnie		(ReadPot i Read4) Wartości wejściowe muszą być niezmodyfikowanymi odczytami potencjometru przez Read4Pots lub inną funkcję, które zwracają wartości niezmodyfikowane
04	dl = pot #	al. = 0 jeśli nie skalibrowane, 1 – jeśli skalibrowane	TestsPotCalibrate. Sprawdza aby zobaczyć czy określony potencjometr jest już skalibrowany. Zwraca stosowną wartość w al. oznaczającą stan kalibracji dla określonego potencjometru.
05	dl = pot #	ax = wartość niezmodyfikowana	ReadRaw. Odczytuje wartość niezmodyfikowaną dla określonego potencjometru. Możemy użyć tej funkcji do pobrania wartości niezmodyfikowanych wymaganych przez podprogram kalibracji.
08	dl = przełącznik#	ax = wartość przełącznika	ReadSw. Odczyt określonego przełącznika i zwraca zero (przełącznik włączony) lub jeden (przełącznik w dole) w rejestrze ax
09		ax = wartości przełącznika	Read16Sw. Funkcja ta pozwala aplikacji odczytu do 16 przełączników sterownika gier w czasie. Bit zero ax odpowiada przełącznika zero, bit 15 ax odpowiada przełącznikowi 15
80h			Remove. Funkcja ta usuwa sterownik z pamięci. Aplikacja generalnie nie wykonują tej funkcji
81h			TestPresence. Ten podprogram zwraca zero w rejestrze ax jeśli sterownik SGDI jest obecny w pamięci. Zwraca wartość w ax niezmienioną w przeciwnym razie (w szczególności, ah będzie jeszcze zawierał 84h)

Tablica 87: Funkcje i API SGDI (int 15h, ah = 84h)

24.5.2 READ4SW

Wejście: ah = 84h, dx =0

Jest to standardowa funkcja BIOS odczytu przełączników. Zwraca stan przełączników od zera do trzech w joysticku w górnych czterech bitów rejestru al. Bit cztery odpowiada przełącznikowi zero, bit pięć przełącznikowi jeden, bit sześć przełącznikowi dwa a bit siedem przełącznikowi trzy. Bit zero oznacza przełącznik w dole, bit jeden odpowiada przełącznikowi w górnej pozycji. Funkcja ta jest dostarczana dla kompatybilności z istniejącymi podprogramami joysticka BIOS. Do odczyt przełączników joysticka powinniśmy użyć funkcji Read16Sw opisanej później w tym dokumencie.

24.5.3 READ4POTS

Wejście: ah = 84h, dx =1

Jest to standardowa funkcja odczytu potencjometrów BIOS. Odczytuje cztery potencjometry w standardowej karcie złącza gier i zwraca ich odczyt w rejestrach ax (oś x /potencjometr 0), bx (oś y / potencjometru 1), cx (potencjometr 2) i dx (potencjometru 3). Są to niezmodyfikowane, nieskalibrowane odczyty potencjometrów , których wartości będą się różnić od maszyny do maszyny i zastosowanej karty I/O/ .Ta funkcja jest dostarczona dla kompatybilności z istniejącymi podprogramami joysticka BIOS. Do odczytu potencjometrów powinniśmy użyć podprogramów ReadPot, Read4 lub ReadRaw opisanych w następnych kilku sekcjach.

24.5.4 READPOT

Wejście: ah =84h, dh =1, dl = numer potencjometru

Odczytuje określony potencjometr i zwraca znormalizowaną wartość potencjometru w zakresie 0...255 w rejestrze al. Podprogram ten ustawia również ah na zero. Chociaż standard SGDI uwzględnia do 255 różnych potencjometrów, większość rozszerzeń tylko wspiera potencjometry zero, jeden, dwa i trzy. Jeśli spróbujemy odczytać potencjometr nie uwzględniony funkcja ta zwróci zero w ah. Ponieważ wartości są znormalizowane, funkcja ta zwraca porównywalne wartości dla danych ustawień sterownika gier bez względu na maszynę, częstotliwość zegara lub karty gier I/O. Na przykład odczyt 128 odpowiadający (odpowiednio) ustawieniu centralnemu na prawie każdej maszynie. Aby osiągnąć wyniki znormalizowane, musimy skalibrować dany potencjometr przed wykonaniem tej funkcji. Zobacz podprogram CalibratePot po więcej szczegółów.

24.5.5 RAED4

Wejście: ah = 84h, al. = maska potencjometru, dx =0200h

Podprogram ten odczytuje cztery potencjometry na karcie rozszerzeń gier, podobnie jak funkcja BIOS (Read4Pots). Jednakże, zwraca wartości znormalizowane w al. (oś x / potencjometr 0), ah (oś y / potencjometr 1), dl (potencjometr 2) i dh (potencjometr 3). Ponieważ ten podprogram zwraca wartości znormalizowane pomiędzy zero i 255, musimy skalibrować potencjometry przed wywołaniem tego kodu. Rejestr al. zawiera wartość „maski potencjometru". Najmniej znaczące cztery bity al. określają czy ten podprogram będzie rzeczywiście odczytywał każdy potencjometr. Jeśli bit zero, jeden, dwa i trzy są jedynkami, wtedy ta funkcja będzie odczytywała odpowiedni potencjometr; jeśli bity te są zerami, podprogram ten nie będzie odczytywał odpowiedniego potencjometru i zwróci zero do odpowiedniego rejestru.

24.5.6 CALIBRATEPOT

Wejście: ah = 84h, dh = 3, dl = potencjometr #, al. = wartość minimalna, bx = wartość maksymalna cx= wartość centralna.

Zanim spróbujemy odczytać potencjometr podprogramem ReadPot lub Read4, musimy skalibrować ten potencjometr. Jeśli odczytamy potencjometr bez uprzedniej kalibracji, sterownik SGDI zwróci tylko zero dla odczytu tego potencjometru. Do kalibracji potencjometru będziemy musieli odczytać wartość niezmodyfikowaną dla tego potencjometru w pozycji minimalnej, maksymalnej i centralnej. To musi być odczyt potencjometru niezmodyfikowany. Używamy odczytu uzyskanego z podprogramu Read4Pots. Teoretycznie musimy tylko skalibrować potencjometr tylko po załadowaniu sterownika SGDI. Jednakże temperatura fluktuacji i prąd analogowych obwodów elektrycznych mogą zdekalibrować potencjometr po istotny zastosowaniu. Dlatego też powinniśmy zdekalibrować potencjometry zmierzając do odczytu za każdym razem, kiedy użytkownik uruchamia swoją aplikację. Co więcej powinniśmy dać użytkownikowi opcję rekalibrację potencjometrów wewnątrz naszego programu.

25.5.7 TESTPOTCALIBRAION

Wejście: ah =84h, dh =4, dl =potencjometr #

Podprogram ten zwraca zero lub jeden w ax, oznaczając odpowiednio nie skalibrowany lub skalibrowany. Możemy użyć tej funkcji aby zobaczyć czy potencjometr jaki zamierzamy użyć, był skalibrowany i możemy przeskoczyć fazę kalibracji. Proszę jednak odnotować komentarz o prądzie w poprzednim paragrafie.

24.5.8 READRAW

Wejście: ah = 84h, dh =5, dl = potencjometr #

Odczytuje określony potencjometr i zwraca niezmodyfikowaną (nie skalibrowaną) wartość w ax. Możemy użyć tego podprogramu dla uzyskania wartości minimalnej, centralnej i maksymalnej do zastosowania przy wywołaniu podprogramu kalibracji.

24.5.9 READSWITCH

Wejście: ah = 84h, dh =8, dl = przełącznik #

Ten podprogram odczytuje określony przełącznik i zwraca zero w ax jeśli przełącznik nie jest w dole. Zwraca jeden jeśli przełącznik jest w dole. Zauważmy, że ta wartość jest w przeciwieństwie do ustawień bitów zwracanych przez funkcję Read4Sw.

Jeśli próbujemy odczytać numer przełącznika dla wejścia, które nie jest dostępne w bieżącym urządzeniu, sterownik SGDI zwróci zero (przełącznik w górze). Standardowe sterowniki gier wspierają tylko przełączniki od zera od trzech a większość joysticków dostarcza tylko dwóch przełączników Dlatego też, o ile chcemy połączyć naszą aplikację z określonym urządzeniem, nie powinniśmy używać innych przełączników niż zero lub jeden.

24.5.10 READ16SW

Wejście: ah =84h, dh =9

Ten podprogram SGDI odczytuje do szesnastu przełączników w pojedynczym wywołaniu. Zwraca wektora bitu w rejestrze ax z bitem zero odpowiadającym przełącznikowi zero, bit jeden odpowiada

przełącznikowi jeden itd. Jedynki oznaczają przełączniki w dole a zera oznaczają przełączniki nie w dole. Ponieważ standardowe złącze gier wspiera tylko cztery przełączniki, tylko bity od zera do trzech al. zawierają znaczące dane (dla tych urządzeń). Wszystkie inne bity będą zawsze zawierały zero. Sterowniki SGDI dla joysticków CH Product's Flightstick Pro i Thrustmaster będą zwracały bity dla całego zbioru dostępnych przełączników w tych urządzeniach.

24.5.11 REMOVE

Wejście: ah =84h, dh =80h

Funkcja ta próbuje usunąć sterownik SGDI z pamięci. Generalnie, tylko sam kod SGDI.EXE wywoływał by ten podprogram. Powinniśmy użyć podprogram TestPresence (opisany poniżej) aby zobaczyć czy sterownik był w rzeczywistości usunięty z pamięci przez tą funkcję.

24.5.12 TESTPRESENCE

Wejście: ah =84h, dh =81h

Jeśli sterownik SGDI jest obecny w pamięci, podprogram ten zwróci ax =0 a wskaźnik do ciągu identyfikującego w es:bx. Jeśli sterownik SGDI nie jest obecny, funkcja ta będzie zwracała ax niezmienione.

24.5.13 STEROWNIK SGDI DLA STANDARDOWEJ KARTY ROZSZERZEŃ GIER

Jeśli piszemy program wykorzystujący funkcje SGDI, odkryjemy ,że funkcja TestPresence będzie prawdopodobnie zwracała „nieobecność" kiedy nasz program poszukuje obecnego sterownika SGDI w pamięci.

Kod asemblerowy który pojawia się na końcu tej sekcji dostarcza pełnej funkcjonalności, bezpłatnych sterowników SGDI dla standardowej karty rozszerzeń gier (kolejna sekcja przedstawia sterownik SGDI dla CH Products Flightstick Pro). Pozwala to nam napisać naszą aplikację korzystającą tylko z funkcji SGDI. Przez wsparcie TSR SGDI z naszym produktem, nasz odbiorca może użyć naszego oprogramowania ze standardowymi joystickami. Później, jeśli nabędzie określone urządzenie ze swoim własnym sterownikiem SGDI, nasze oprogramowanie będzie automatycznie działał z tym sterownikiem nie zmieniając naszego oprogramowania.

Jeśli nie chcemy aby użytkownik uruchamiał TSR przed naszą aplikacją, możemy zawsze wprowadzić poniższy kod wewnątrz naszego kodu programu i aktywować go jeśli funkcja SGDI TestPresence określi ,że żaden inny sterownik SGDI nie jest obecny w pamięci kiedy startujemy nasz program

Tu mamy kompletny kod dla sterownika SGDI standardowego złącza gier:

```
                .286
                page    58, 132
                name    SGDI
                title   SGDI Driver for Standard Game Adapter Card
                subttl  Ten program jest Public Domain

; SGDI.EXE
;
;       Usage:
;               SDGI
;
; Program ten ładuje TSR, który aktualizuje INT 15 więc przypadkowy program może odczytać joystick
; w przenośny sposób.
;
; Musimy załadować cseg w pamięci przed każdym innymi segmentami!

cseg            segment para  public 'code'
cseg            ends

; Kod inicjalizujący, którego nie potrzebujemy z wyjątkiem początkowego ładowania, idzie w poniższym
; segmencie:

; Initialize    segment para public 'INIT'
```

```
;Iinitialize     ends

; Podprogramy Biblioteki Standardowej ,które zostaną pokazane później
                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list

sseg            segment para stack 'stack'
sseg            ends

zzzzzzseg       segment para public 'zzzzzzseg'
zzzzzzseg       ends

CSEG            segment para public 'CODE'
                assume  cs:cseg, ds:nothing

wp              equ     <word ptr>
byp             equ     <byte ptr>

Int15Vect       dword   0
PSP             word    ?

; Adresy portu dla typowego złącza joysticka:

JoyPort         equ     201h
JoyTrigger      equ     201h

; Struktura danej przechowująca informacje o każdym potencjometrze (głownie do celów kalibracji i
; normalizacji)

Pot             struc
PotMask         byte    0               ;maska potencjometru  dla sprzętu
DidCal          byte    0               ;czy ten potencjometr jest skalibrowany?
min             word    5000            ;minimalna wartość potencjometru
max             word    0               ;maksymalna wartość potencjometru
center          word    0               ;wartość potencjometru po środku
Pot             ends

; Zmienne dla każdego potencjometru. Musimy zainicjalizować maski więc maskujemy wszystkie bity z
; wyjątkiem przychodzącego bitu dla każdego potencjometru

Pot0            Pot     <1>
Pot1            Pot     <2>
Pot2            Pot     <4>
Pot3            Pot     <8>

; IDString  pobiera adres przekazywany z powrotem do kodu wywołującego w funkcji testpresence. Cztery
; bajty przed IDString muszą zawierać numer seryjny i bieżąca liczbę sterownika

SerialNumber    byte    0,0,0
IDNumber        byte    0
IDString        byte    "Standard SGDI Driver", 0
                byte    "Public Domain Driver Written by Randall L. Hyde", 0
;=============================================================================
;
; ReadPots-     AH zawiera bit maski określający , który potencjometr będzie czytany. Bit 0 ma wartość jeden
;               kiedy powinniśmy odczytać potencjometr 0, bit 1 jest jedynką jeśli powinniśmy odczytać
;               potencjometr 1, bit 2 jest jedynką jeśli powinniśmy odczytać potencjometr 2, bit 3 jest jedynką
```

```
;                   jeśli powinniśmy odczytać potencjometr 3. Wszystkie inne bity powinny być zerami.
;
;           Kod ten zwraca wartości potencjometrów w SI, BX, BP i DI dla potencjometrów 0,1,2 i 3
;
ReadPots        proc    near
                sub     bp, bp
                mov     si, bp
                mov     di, bp
                mov     bx, bp

; Oczekujemy na poprzedni sygnał kończący przed próbą odczytu tego potencjometru .Jest możliwe, że ostatni
; potencjometr jaki odczytaliśmy był bardzo krótki. Jednakże, wyzwolenie sygnału startowego timera działa dla
; wszystkich czterech potencjometrów .Kod ten kończy jeśli tylko skończy się czas bieżącego potencjometru
; Jeśli użytkownik bezpośrednio odczyta inny potencjometr, jest całkiem możliwe, że nowy czas potencjometru
; nie upłynie z poprzedniego odczytu. Poniższa pętla upewnia nas , że nie mierzymy czasu z poprzedniego
; odczytu.

                mov     dx, JoyPort
                mov     cx, 400h
Wait4Clean      in      al, dx
                and     al, 0Fh
                loopnz  Wait4Clean
;Okay, odczytano potencjometry. Poniższy kod wyzwala chip timera 558 a potem umieszcza w pętli dopóki
; wszystkie cztery bity potencjometrów (zamaskowane maską potencjometru w AL.) nie staną się zerami. Za
; każdym razem podczas tej pętli jeśli jeden lub więcej z tych bitów zawiera zero, pętla ta zwiększa odpowiedni
; rejestr(-y)

                mov     dx, JoyTrigger
                out     dx, al.                 ;wyzwolenie potencjometru
                mov     dx, JoyPort
                mov     cx, 1000h
PortReadLoop:   in      al, dx
                and     al, ah
                jz      potReadDone
                shr     al, 1
                adc     si, 0                   ;zwiększamy SI jeśli potencjometr 0 jest aktywny
                shr     al., 1
                adc     bx, 0                   ;zwiększamy BX jeśli potencjometr 1 jest aktywny
                shr     al., 1
                adc     bp, 0                   ;zwiększamy bp jeśli potencjometr 2 jest aktywny
                shr     al., 1
                adc     di, 0                   ;zwiększamy DI jeśli potencjometr 3 jest aktywny
                loop    PotReadLoop

                and     si, 0FFFh               ;Jeśli dojdziemy do tego punktu, jeden lub więcej
                and     bx, 0FFFh               ;potencjometrów przekroczy czas oczekiwania (
                and     bp, 0FFFh               ;dlatego, że zazwyczaj nie są podłączone. Rejestr
                and     di, 0FFFh               ;zawiera 4000h, ustawiamy go na 0
PotReadDone:    ret
ReadPots        endp

;-------------------------------------------------------------------------------------------------------------------------
-
;
; Normalize-    BX zawiera wskaźnik do struktury potencjometru, AX zawiera wartość potencjometru.
;               Normalizujemy tą wartość zgodnie ze skalibrowanym potencjometrem.
;
; Notka: DS. musi wskazywać cseg przed wywołaniem tego podprogramu
```

```
                assume  ds.:  cseg
Normalize       proc    near
                push    cx

; Na zdrowy rozum, upewniamy się czy proces kalibracji przeszedł okay

                cmp     [bx].Pot.DidCal, 0      ;czy potencjometr jest skalibrowany?
                je      BadNorm                 ;jeśli nie wychodzimy

                mov     dx, [bx].Pot.Center     ; wykonujemy sprawdzenie wartości minimalnej
                cmp     dx, [bx].Pot.Min        ;centralnej i maksymalnej aby upewnić się , że
                jbe     BadNorm                 ; min < center <max
                cmp     dx, [bx].Pot.Max
                jae     BadNorm

; Obcinamy wartość jeśli jest poza zakresem

                cmp     ax, [bx].Pot.Min        ;jeśli wartość jest mniejsza niż wartość minimum
                ja      MinOkay                 ;ustawiamy na wartość minimalną
                mov     ax, [bx].Pot.Min

MinOkay:
                cmp     ax, [bx].Pot.Max        ;jeśli wartość jest większa niż wartość maksymalna
                jb      MaxOkay                 ;ustawiamy wartość maksymalną
                mov     ax, [bx].Pot.Max

MaxOkay:

; Wyskalujemy to wokół centrum:

                cmp     ax, [bx].Pot.Center     ;zobaczymy czy jest mniejsza lub większa
                jb      Lower128                ;od wartości centralnej

;Okay, bieżący odczyt jest większy niż wartość centralna, skalujemy odczyt do zakresu 128...255:

                sub     ax, [bx].Pot.Center
                mov     dl, ah                          ;mnożymy przez 128
                mov     ah, al.
                mov     dh, 0
                mov     al, dh
                shr     dl, 1
                rcr     ax, 1
                mov     cx, [bx].Pot.Max
                sub     cx, [bx].Pot.Center
                jz      BadNorm                 ;Zabezpieczenie przed dzieleniem przez zero
                div     cx                      ;obliczamy wartość znormalizowaną
                add     ax, 128                 ;skalujemy zakres 128...255
                cmp     ah, 0
                je      NormDone
                mov     ax, 0ffh                ;wynik musi mieścić się w 8 bitach!
                jmp     NormDone

; Jeśli odczyt jest poniżej wartości centralnej, skalujemy ją do zakresu 0..127:

Lower128:       sub     ax, [bx].Pot.Min
                mov     dl, ah
                mov     ah, al
                mov     dh, 0
                mov     al, dh
```

```
                shr     dl, 1
                rcr     ax, 1
                mov     cx, [bx].Pot.Center
                sub     cx, [bx].Pot.Min
                jz      BadNorm
                div     cx
                cmp     ah, 0
                je      NormDone
                mov     ax, 0ffh
                jmp     NormDone

;Jeśli coś poszło źle , zwracamy zero jako wartość znormalizowaną

BadNorm:        sub     ax, ax

NormDone        pop     cx
                ret
Normalize       endp
                assume  ds:nothing

;=============================================================================
; Funkcje obsługi INT 15h
;=============================================================================
;
; Chociaż są zdefiniowane jaki bliskie procedury, nie są w rzeczywistości procedurami. Kod MyInt15 skacze do
; każdej z nich z BX, daleki adres powrotu i  flagi są usytuowane na stosie. Każdy z tych podprogramów musi
; obsłużyć właściwie stos.
;-----------------------------------------------------------------------------------------------------------------------
-
;
;BIOS-  Obsługuje dwie funkcje BIOS, DL =0 odczyt przełączników, DL =1 odczyt potencjometrów. Dla
;       podprogramów BIOS, zignorujemy cooley switch i po prostu odczytujemy inne cztery przełączniki
;

BIOS            proc    near
                cmp     dl, 1                   ;zobacz czy program przełącznika czy potencjometru
                jb      Read4Sw
                je      ReadBIOSPots

; Jeśli nie poprawna funkcja BIOS, skok do oryginalnego programu obsługi INT 15h i zezwolenie na zajęcie się
; tą funkcją

                pop     bx
                jmp     cs: Int15Vect           ;pozwólmy na obsłużenie go!

;BIOS odczytuje funkcję przełącznika

Read4Sw:        push    dx
                mov     dx, JoyPort
                in      al, dx
                and     al, 0F0h                ;zwracamy tylko wartości przełączników
                pop     dx
                pop     bx
                iret

; Odczyt funkcji potencjometrów BIOS

ReadBIOSPots:   pop     bx                      ;wartość zwracana w BX!
                push    si
```

```
                push    di
                push    bp
                mov     ah, 0Fh                 ;odczyt wszystkich czterech potencjometrów
                call    ReadPots
                mov     ax, si
                mov     cx, bp                  ;BX już  zawiera odczyt z potencjometru 1
                mov     dx, di
                pop     bp
                pop     di
                pop     si
                iret
BIOS            endp

;-----------------------------------------------------------------------------------------------------------------------
-
; ReadPot-      Na wejściu DL zawiera numer potencjometru do odczytu. Odczyt i normalizacja tego
;               potencjometru i zwracany wynik w AL.
;

                assume  ds.: cseg
ReadPot         proc    near
;;;;;;;;;;;;;;  push    bx                              ;Już na stosie
                push    ds.
                push    cx
                push    dx
                push    si
                push    di
                push    bp

                mov     bx, cseg
                mov     ds., bx
; Jeśli dl = 0, odczyt i normalizacja wartości dla potencjometru 0, jeśli nie , próbujemy jakiś inny potencjometr

                cmp     dl, 0
                jne     Try1
                mov     ah, Pot0.PotMask        ;pobranie bitu dla tego potencjometru
                call    ReadPots                ;odczyt potencjometru 0
                lea     bx, Pot0                ;wskaźnik do danego potencjometru
                mov     ax, si                  ;pobranie odczytu potencjometru 0
                call    Normalize               ;normalizacja od 0 do FFh
                jmp     GotPot                  ;powrót do kodu wywołującego

; Test dla DL =1 (odczyt i normalizacja potencjometru 1)

Try1:           cmp     dl, 1
                jne     Try2
                mov     ah, Pot1.PotMask
                call    ReadPots
                mov     ax, bx
                lea     bx, pot1
                call    Normalize
                jmp     GotPot

;Test dla DL=2 (odczyt I normalizacja potencjometru 2)

Try2:           cmp     dl, 2
                jne     Try3
                mov     ah, Pot2.PotMask
                call    RaedPots
```

```
                lea     bx, Pot2
                mov     ax, bp
                call    Normalize
                jmp     GotPot

;Test dla DL=3 (odczyt I normalizacja potencjometru 3)

Try3:           cmp     dl, 3
                jne     BadPot
                mov     ah, Pot3.PotMask
                call    ReadPots
                lea     bx,  Pot3
                mov     ax, di
                call    Normalize
                jmp     GotPot
; Złą wartość w DL jeśli doszliśmy do tego miejsca. Standardowe złącze gier wspiera tylko cztery
; potencjometry.

BadPot:         sub     ax, ax                  ;Niedostępny potencjometr, zwracane zero
GotPot:         pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     ds.
                pop     bx
                iret
ReadPot         endp
                assume  ds.:nothing

;-------------------------------------------------------------------------------------------------------------------------
-;
; ReadRaw-      Na wejściu DL zawiera numer potencjometru do odczytu. Odczytuje ten potencjometr i zwraca
;               nieznormalizowany wynik w AX

                assume  ds:cseg

ReadRaw         proc    near
;;;;;;;;;;;;;;;;;       push    bx                  ;Już na stosie
                push    ds.
                push    cx
                push    dx
                push    si
                push    di
                push    bp

                mov     bx, cseg
                mov     ds., bx

;kod ten jest prawie identyczny z kodem ReadPot. Jedyna różnica jest taka, że nie musimy martwić się
; normalizacją wyniku i (oczywiście) zwracamy wartość w AX zamiast w Al.


                cmp     dl, 0
                jne     Try1
                mov     ah, Pot0.PotMask
                call    ReadPots
                mov     ax, si
                jmp     GotPot
```

```
Try1:           cmp     dl, 1
                jne     Try2
                mov     ah, Pot1.PotMask
                call    ReadPots
                mov     ax, bx
                jmp     GotPot

Try2:           cmp     dl, 2
                je      Try3
                mov     ah., Pot2.PotMask
                call    ReadPots
                mov     ax, bp
                jmp     GotPot

Try3:           cmp     dl, 3
                jne     BadPot
                mov     ah, Pot3.PotMask
                call    ReadPots
                mov     ax, di
                jmp     GotPot

BadPot:         sub     ax, ax                  ;Potencjometr niedostępny, zwracane zero
GotPot:         pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     ds.
                pop     bx
                iret
ReadRaw         endp
                assume  ds:nothing

;-----------------------------------------------------------------------------------------------------------------------
-
; Read4Pots-    Odczytuje potencjometr zero, jeden , dwa i trzy zwracając ich wartości w AL ,AH ,DL i DH
;
;               Na wejściu, AL zawiera maskę potencjometru dla wyboru, który potencjometr powinniśmy
;               odczytać (bit 0 =1  dla potencjometru 0, bit 1=1 dla potencjometru 1 itd.)
;

Read4Pots       proc    near
;;;;;;;;;;;;;;;;;       push    bx                      ;już na stosie
                push    ds.
                push    cx
                push    si
                push    di
                push    bp

                mov     dx, cseg
                mov     ds., dx
                mov     ah, al.
                call    ReadPots

                push    bx                      ;zachowanie odczytu potencjometru 1
                mov     ax, si                  ;pobranie odczytu potencjometru 0
                lea     bx, Pot0                ;bx wskazuje na potencjometr 0
                call    Normalize               ;normalizacja
```

```
                mov     cl, al.                 ;zachowanie na później

                pop     ax                      ;odzyskanie odczytu potencjometru 1
                lea     bx, Pot1
                call    Normalize
                mov     ch, al.                 ;zachowanie znormalizowanej wartości

                mov     ax, bp
                lea     bx, Pot2
                call    Normalize
                mov     dl, al.                 ;wartoœæ Pot2

                mov     ax, di
                lea     bx, Pot3
                call    Normalize
                mov     dh, al.                 ;wartość Pot3
                mov     ax, cx                  ;potencjometr 0 I 1

                pop     bp
                pop     di
                pop     si
                pop     cx
                pop     ds.
                pop     bx
                iret
Read4Pots       endp

;-------------------------------------------------------------------------------------------------------------------------------
-
; CalPot-       Kalibracja potencjometru określonego przez DL. Na wejściu AL. zawiera minimalną
;               wartość potencjometru (lepiej żeby była mniejsza niż 256!), BX zawiera maksymalną wartość
;               potencjometru a CX zawiera centralną wartość potencjometru
;
                assume  ds.:cseg

CalPot          proc    near
                pop     bx                              ;odzyskanie wartości maksymalnej
                push    ds.
                push    si
                mov     si, cseg
                mov     ds., si

; Sprawdzenie parametrów, sortowanie według rosnącego porządku:

                mov     ah, 0
                cmp     bx, cx                  ;upewnijmy się ,że center < max
                ja      GoodMax
                xchg    bx, cx
GoodMax:        cmp     ax, cx                  ;upewnijmy się, że min < center
                jb      GoodMin                 ;(notka: może  okazać się center < max)
                xchg    ax, cx
GoodMin:        cmp     cx, bx                  ;ponownie pewność ,że center < max
                jb      GoodCenter
                xchg    cx, bx
GoodCenter:

;Okay, domyślamy się co skalibrować;

                lea     si, Pot0
```

```
                cmp     dl, 1
                jb      DoCal                   ;skok jeśli jest to potencjometr 0
                lea     si, Pot1
                je      DoCal                   ;skok jeśli jest to potencjometr 1
                lea     si, Pot2
                cmp     dl, 3
                jb      DoCal                   ;skok jeśli jest to potencjometr 2
                jne     CalDone                 ;skok jeśli nie jest to potencjometr 3
                lea     si, Pot3

DoCal:          mov     [si].Pot.Min, ax        ; przechowujemy wartości minimum, maksimum i
                mov     [si].Pot.Max,bx         ; centrum.
                mov     [si].Pot.Center, cs
                mov     [si[.Pot,DidDal, 1
CalDone:        pop     si
                pop     ds
                iret
CalPot          endp
                assume  ds:nothing

;-----------------------------------------------------------------------------------------------------------------------
-
; TestCal-      Sprawdzamy czy określony potencjometr przez DL został już kalibrowany
;
                assume  ds.:cseg
TestCal         proc    near
;;;;;;;;;;;;;   push    bx                      ;już na stosie
                push    ds.
                mov     bx, cseg
                mov     ds., bx

                sub     ax, ax                  ;zakładamy, że nie było kalibracji (również zero w AH)
                lea     bx, Pot0                ;pobranie adresu określonej struktury danych
potencjometru
                cmp     dl, 1                   ; do rejestru BX
                jb      GetCal
                lea     bx, Pot1
                je      GetCal
                lea     bx, Pot2
                cmp     dl, 3
                jb      GetCal
                jne     BadCal
                lea     bx, Pot3

GetCal:         mov     al., [bx].Pot.DidCal
BadCal:         pop     ds
                pop     bx
                iret
TestCal         endp
                assume  ds:nothing

;-----------------------------------------------------------------------------------------------------------------------
-
;
; ReadSw-       Odczyt przełącznika, którego numer pojawia się w DL

ReadSw          proc    near
;;;;;;;;;;;;;;  push    bx                      ;już na stosie
                push    cx
```

```
                sub     ax, ax                  ;założenie ,że  brak takiego przełącznika
                cmp     dl, 3                   ;zwracane jeśli numer przełącznika jest większa niż
                ja      NotDown                 ; trzy

                mov     cl, dl                  ;zachowanie przełącznika do odczytu
                add     cl, 4                   ;przesunięcie z pozycji cztery do zero
                mov     dx, JoyPort
                in      al., dx                 ;odczyt przełącznika
                shr     al., cl                 ;przesuwa żądany bit przełącznika do bitu 0
                xor     al., 1                  ;odwrócenie w dół = 1
                and     ax, 1                   ;usunięcie innego niepotrzebnych bitów
NotDown:        pop     cx
                pop     bx
                iret
ReadSw          endp

;-------------------------------------------------------------------------------------------------------------------------------
-
;
; Read16Sw-     Odczyt wszystkich czterech przełączników i zwrot ich wartości w AX
Read16Sw        proc    near
;;;;;;;;;;;;;;;;;;      push    bx                              ;już na stosie
                mov     dx, JoyPort
                in      al., dx
                shr     al., 4
                xor     al., 0Fh                        ;odwrócenie wszystkich przełączników
                and     ax, 0Fh                         ;ustawienie pozostałych bitów na zero
                pop     bx
                iret
Read16Sw        endp

;*******************************************************************************************
*;
; MyInt15-      Aktualizacja podprogramu BIOS INT 15 dla sterowania odczytem joysticka
;

MyInt15         proc    far
                push    bx
                cmp     ah, 84h                         ;kod joysticka
                je      DoJoystick
OtherInt15:     pop     bx
                jmp     cs: Int15Vect

DoJoystick:     mov     bh, 0
                mov     bl, dh
                cmp     bl, 80h
                jae     VendorCalls
                cmp     bx, JmpSize
                jae     OtherInt15
                shl     bx, 1
                jmp     wp  cs: jmptable[bx]

jmptable        word    BIOS
                word    ReadPot, Read4Pots, CalPot, TestCal
                word    ReadRaw, OtherInt15, OtherU\Int15
                word    ReadSw, Read16Sw
JmpSize         =       ($-jmptable)/2
```

```
; Obsługa  określonej funkcji

VendorCalls:    je      RemoveDriver
                cmp     bl, 81h
                je      TestPresence
                pop     bx
                jmp     cs: Int15Vect

; TestPresence-   Zwraca zero w AX I wskazuje na ID ciągu w ES:BX

TestPresence:   pop     bx                      ;pobranie starej wartości ze stosu
                sub     ax, ax
                mov     bs, cseg
                mov     es, bx
                lea     bx, IDString
                iret

; RemoveDriver- Jeśli nie ma innych sterowników załadowanych po tym w pamięci, wyłącza go
;               i usuwa z pamięci

;RemoveDriver:
                push    ds.
                push    es
                push    ax
                push    dx

                mov     dx, cseg
                mov     ds., dx

;zobaczmy czy zaktualizowaliśmy ostatni podprogram INT 15h

                mov     ax, 3515h
                int     21h
                cmp     bx, offset MyInt15
                jne     CantRemove
                mov     bx, es
                cmp     bx, wp  seg MyInt15
                jne     CantRemove

                mov     ax, PSP                 ;zwolnienie pamięci
                mov     es, ax
                push    es
                mov     ax, es:[2ch]            ;najpierw zwalniamy blok środowiska
                mov     es, ax
                mov     ah, 49h
                int     21h
                pop     es                      ;teraz zwalniamy przestrzeń programu
                mov     ah, 49h
                int     21h

                lds     dx, Int15Vect           ;przywrócenie poprzedniego wektora
                mov     ax, 2515h
                int     21h

CantRemove:      pop    dx
                pop     ax
                pop     es
                pop     ds
                pop     bx
```

```
                iret
MyInt15         endp
cseg            ends

Initialize      segment para  public  'INIT'
                assume  cs: Initialize, ds:cseg
Main            proc
                mov     ax, cseg                        ;pobranie wskaźnika do segmentu zmiennych
                mov     es, ax
                mov     es:PSP, ds.                     ;zachowanie wartości PSP
                mov     ds, ax

                mov     ax, zzzzzzseg
                mov     es, ax
                mov     cx, 100h
                meminit2

                print
                byte    „Standard Game Device Interface driver", cr, lf
                byte    "PC Compatible Game Adapter Cards ", cr, lf
                byte    "Written by Randall Hyde", cr, lf
                byte    cr, lf
                byte    cr, lf
                byte    "'SGDI REMOVE' usuwa sterownik z pamięci", cr, lf
                byte    lf
                byte    0

                mov     ax, 1
                argv                                    ;jeśli nie ma parametrów pusty ciąg
                stricmpl
                byte    „REMOVE", 0
                jne     NoRmv
                mov     dh, 81h                         ;usuwanie opcodu
                mov     ax, 84ffh
                int     15h                             ;zobaczmy czy wszystko już załadowane
                test    ax, ax                          ;pobranie zera?
                jz      Installed
                print
                byte    „Sterownik SGDI nie jest obecny w pamięci, polecenie"
                byte    „ REMOVE jest ignorowane", cr, lf, 0
                mov     ax, 4c01h                                   ;wyjście do DOS'a
                int     21h

Installed:      mov     ax, 8400h
                mov     dh, 80h                                     ;funkcja usuwająca
                int     15h
                mov     ax, 8400h
                mov     dh, 81h                                     ;wywołanie TestPresence
                int     15h
                cmp     ax, 0
                je      NotRemoved
                print
                byte    „Usunięcie sterownika SGDI z pamięci zakończone powodzeniem"
                byte    cr, lf, 0
                mov     ax, 4c00h                               ;wyjście do DOS
                int     21h

NotRemoved:     print
                byte    „Sterownik SGDI jest nadal obecny w pamięci.", cr, lf,0
```

```
                mov     ax, 4c01h                       ;wyjście do DOS
                int     21h

;Okay, aktualizujemy INT 15

NoRmv:
                mov     ax, 3515h
                int     21h
                mov     wp  Int15Vect, bx
                mov     wp  Int15Vect+2, es

                mov     dx, cseg
                mov     ds, dx
                mov     dx, offset MyInt15
                mov     ax, 2515h
                int     21h

                mov     dx, cseg
                mov     ds, dx
                mov     dx,  seg  Initiazlize
                sub     dx, ds:psp
                add     dx, 2
                mov     ax, 3100h
                int     21h
Main            endp

Iniyialize      ends

seg             segment para  stack  'stack'
                word    128  dup  (0)
endstk          word    ?
sseg            ends

zzzzzzseg       segment para  public 'zzzzzzseg'
                byte    16  dup  (0)
zzzzzzseg       ends
                end     Main
```

Poniższy program wykonuje kilka różnych typów wywołań sterownika SGDI. Możemy użyć go do przetestowania SGDI TSR:

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

cseg            segment para public 'code'
                assume  cs:cseg, ds:nothing

MinVal0         word    ?
MinVal1         word    ?
MaxVal0         word    ?
MaxVal1         word    ?

;Wait4Button-   Oczekuje dopóki użytkownik nie naciśnie i zwolni przycisku

Wait4Button     proc    near
                push    ax
                push    dx
```

```
                push    cx

W4BLp:          mov     ah, 84h
                mov     dx, 900h            ;odczyt mnij znaczących 16 przycisków
                int     15h
                cmp     ax, 0               ;jakiś przycisk w dole? Jeśli nie pętla dopóki jest
                je      W4BLp

                xor     cx, cx              ;opóźnienie pętli
Delay:          loop    Delay
W4nBLp:         mov     ah, 84h             ;teraz czekamy aż użytkownik zwolni wszystkie przyciski
                mov     dx, 900h
                int     15h
                cmp     ax, 0
                jne     W4nBLp

Delay2:         loop    Delay2
                pop     cx
                pop     dx
                pop     ax
                ret
Wait4Button     endp

Main            proc
                print
                byte    “SGDI Test Program”, cr, lf
                byte    “Written by Randal Hyde”, cr,lf,lf
                byte    “Press any key to continue”, cr, lf,0

                getc

                mov     ah, 84h
                mov     dh, 4                       ;funkcja Testpresence
                int     15h
                cmp     ax, 0
                je      MainLoop0
                print
                byte    “Żaden sterownik SGDI nie jest obecny w pamięci.”,cr,lf,0
                jmp     Quit

MainLoop0:      print
                byte    „BIOS” 0

;Okay, odczytujemy przełączniki i niezmodyfikowane wartości potencjometrów używając kompatybilnej funkcji
; BIOS

                mov     ah, 84h
                mov     dx, 0               ;kompatybilny z BIOS odczyt przełączników
                int     15h
                puth                        ;wartość wyjściowa przełącznika
                mov     al.,  ‘ ‘
                putc

                mov     ah, 84h             ;kompatybilny z BIOS odczyt potencjometrów
                mov     dx, 1
                int     15h
                putw
                mov     al., ‘ ‘
                putc
```

```
        mov     ax, bx
        putw
        mov     al., ‘ ‘
        putc
        mov     ax, cx
        putw
        mov     al., ‘ ‘
        putc
        mov     ax, dx
        putw

        putcr
        mov     ah, 1                   ;powtarzanie pętli dopóki naciśnięty klawisz
        int     16h
        je      MainLoop0
        getc
```

; Odczyt wartości minimalnej i maksymalnej dla każdego potencjometru od użytkownika, więc można
; skalibrować potencjometry

```
        print
        byte    cr, lf, lf, lf
        byte    “Przesuwamy joystick w lewy górny róg I naciskamy “
        byte    „ jakiś przycisk”,cr,lf,0

        call    Wait4Button
        mov     ah, 84h
        mov     dx, 1                   ;odczyt wartości niezmodyfikowanej
        int     15h
        mov     MinVal0, ax
        mov     MinVal1, bx

        print
        byte    cr, lf
        byte    „Przesuwamy joystick w prawy dolny róg „
        byte    „ i naciskamy jakiś przycisk”,cr,lf,0

        call    Wait4Button
        mov     ah, 84h
        mov     dx, 1                   ;odczyt wartości niezmodyfikowanej
        int     15h

        mov     MaxVal0, ax
        mov     MaxVal1, bx
```

; Kalibracja potencjometrów

```
        mov     ax, MinVal0             ;będzie osiem bitów lub mniej
        mov     bx, MaxVal0
        mov     cx, bx                  ;obliczamy wartość centralną jako średnią
        add     cx, ax                  ; z tych dwóch (jest to niebezpieczne, ale
        shr     cx, 1                   ;zazwyczaj działa!)
        mov     ah, 84h
        mov     dx, 300h                ;kalibracja potencjometru 0
        int     15h

        mov     ax, MinVal1             ;będzie osiem lub mnij bitów
        mov     bx, MaxVal1
        mov     cx, bx                  ;obliczamy wartość centralną jako średnią
```

```
                add     cx, ax                  ;z tych dwóch (jest to niebezpieczne, ale
                shr     cx, 1                   ;zazwyczaj działa!)
                mov     ah, 84h
                mov     dx, 301h                ;kalibrujemy potencjometr 1

MainLoop0:      print
                byte    „ReadSw:”, 0

;Okay, odczytujemy wartości przełączników i niezmodyfikowanych potencjometrów używając kompatybilnych
; funkcji BIOS

                mov     ah, 84h
                mov     dx, 800h                ;odczyt przełącznika zero
                int     15h
                or      al., ‘0’
                putc

                mov     ah, 84h
                mov     dx, 801h                ;odczyt przełącznika jeden
                int     15h
                or      al., ‘0’
                putc

                mov     ah, 84h
                mov     dx, 802h                ;odczyt przełącznika dwa
                int     15h
                or      al., ‘0’
                putc

                mov     ah, 84h
                mov     dx, 803h                ;odczyt przełącznika trzy
                int     15h
                or      al., ‘0’
                putc

                mov     ah, 84h
                mov     dx, 804h                ;odczyt przełącznika cztery
                int     15h
                or      al., ‘0’
                putc

                mov     ah, 84h
                mov     dx, 805h                ;odczyt przełącznika pięć
                int     15h
                or      al., ‘0’
                putc

                mov     ah, 84h
                mov     dx, 806h                ;odczyt przełącznika sześć
                int     15h
                or      al., ‘0’
                putc

                mov     ah, 84h
                mov     dx, 807h                ;odczyt przełącznika siedem
                int     15h                     ;nie będziemy martwić się ich  większą ilością
                or      al., ‘0’
                putc
                mov     al., ‘ ‘
```

```
                putc

                mov     ah, 84h
                mov     dh, 9                   ;odczyt wszystkich 16 przełączników
                int     15h
                putw

                print
                byte    „ Potencjometry: „„ 0
                mov     ax, 8403h               ;odczyt potencjometru joysticka
                mov     dx, 200h                ;odczyt czterech potencjometrów
                int     15h
                puth
                mov     al., ‘ ‘
                putc
                mov     al., ah
                puth
                mov     al., ‘ ‘
                putc

                mov     ah, 84h
                mov     dx, 503h                ;odczyt niezmodyfikowany, potencjometr 3
                int     15h
                putw

                putcr
                mov     ah, 1                   ;powtarzanie pętli dopóki naciśnięty klawisz
                int     16h
                je      MainLoop1
                getc

Quit:           ExitPgm
Main            endp

cseg            ends

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

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

24.6 STEROWNIK SGDI DLA CH PRODUCTS' FLIGHT STICK PRO™

Joystick CH Products' Flight Stick Pro jest dobrym przykładem specjalizowanego produktu dla którego sterownik SGDI jest doskonałym rozwiązaniem. Flight Stick Pro dostarcza trzech potencjometrów i pięciu przełączników, piąty przełącznik stanowi specjalną piątą pozycję **cooley switch** Chociaż potencjometry w Flight Stick Pro mapują do trzech wejść analogowych w standardowej karcie rozszerzeń gier (potencjometr zero, jeden i trzy), jest niewystarczająco wejść cyfrowych do obsługi ośmiu wejść koniecznych dla czterech przycisków i **cooley switch** Flight Stick Pro.

Flight Stick Pro (FSP) używa kilku układów elektronicznych mapujących te osiem pozycji przełącznika do czterech bitów wejściowych. Robiąc to, określają jedno ograniczenie przy stosowaniu przełączników FSP – możemy tylko nacisnąć jeden w jednym czasie. Jeśli przetrzymujemy dwa lub więcej przełączników w tym samym czasie, FSP wybierze jeden z przełączników i zaraportuje tą wartość; zignoruje inne przełączniki dopóki nie zwolnimy przycisku. Ponieważ tylko jeden przełącznik może być odczytany w jednym czasie, FSP wygeneruje wartość czterech bitów, które określą aktualny stan przełączników. Zwraca te

cztery bity jako wartość przełączników w standardowej karcie rozszerzeń gier. Poniższa tablica pokazuje te wartości dla każdego z przełączników:

Wartości (binarnie)	Priorytet	Pozycja przełącznika
0000	Najwyższy	Pozycja górna w **cooley switch**
0100	7	Prawa pozycja w **cooley switch**
1000	6	Dolna pozycja w **cooley switch**
1100	5	Lewa pozycja w **cooley switch**
1110	4	Wyzwolenie joysticka
1101	3	Najbardziej na lewo przycisk joysticka
1011	2	Najbardziej na prawo przycisk joysticka
0111	Najniższy	Środkowy przycisk na joysticku
1111		Żaden przycisk aktualnie nie jest w dole

Tablica 88: Wartości zwraca przełącznika Flight Stick Pro

Zauważmy, że przyciski wyglądają jak pojedyncze naciśnięcie przycisku. Pozycja **Cooley switch** zawiera wartość pozycji w bitach sześć i siedem; bity cztery i pięć zawsze zawierają zero, kiedy **cooley switch** jest aktywny.

Sterownik SGDI dla Flight Stick Pro jest bardzo podobny do standardowej karty rozszerzeń gier sterownika SGDI. Ponieważ Flight Stick Pro dostarcza tylko trzech potencjometrów, kod ten nie zajmuje się próbami odczytu potencjometru 2 (nie istniejący) Oczywiście, przełączniki w FSP różnią się trochę od tych w joysticku standardowym, więc sterownik FSP SGDI mapuje przełączniki FSP do ośmiu logicznych przełączników SGDI. Poprzez odczyt przełączników zero do siedem, możemy przetestować warunki w FSP:

Numer przełącznika SGDI:	Mapowanie do tego przełącznika FSP
0	Wyzwolenie joysticka
1	Lewy przycisk joysticka
2	Środkowy przycisk joysticka
3	Prawy przycisk joysticka
4	Górna pozycja **cooley**
5	Lewa pozycja **cooley**
6	Prawa pozycja **cooley**
7	Dolna pozycja **cooley**

Tablica 89: Mapowanie przełączników Flight Stick Pro

Sterownik FSP SGDI zawiera jedną z nowoczesnych cech, pozwala użytkownikowi zamieniać lewego i prawego przełącznika w joysticku. Wiele gier często przydziela ważne funkcje do spustu i lewego przycisku ponieważ są łatwiejsze do naciśnięcia (prawo ręczny gracz może łatwo nacisnąć lewy przycisk swoim kciukiem). Poprzez wpisanie „LEFT" w lini poleceń, sterownik FSP SGDI zmienimy funkcje lewego i prawego przycisku, aby leworęczny gracz mógł łatwo aktywować tą funkcję kciukiem.

Poniższy kod dostarcza kompletnego listingu dla sterownika FSP SGDI. Zauważmy, że możemy zastosować ten sam program testowy z poprzedniej sekcji dla przetestowania tego sterownika

```
                .286
                page    58, 132
                name    FSPSGDI
                title   FSPSGDI (CH Products Standard Game Device Interface)

;FSPGDI.EXE
;
;       Użycie:
;               FSPSGDI         {LEFT}
;
; Program ten ładuje TSR, który aktualizuje INT 15 więc program może odczytać  joystick CH Products Flight
; Stick w przenośny sposób
```

```
wp              equ     < word ptr >
byp             equ     < byte ptr >

; Musimy załadować cseg do pamięci przed innymi segmentami!

cseg            segment para public ‘code’
cseg            ends

; Kod inicjalizujący

Initialize      segment para public „INIT’
Initialize      ends

; Podprogramy Biblioteki Standardowej UCR, które pojawią się później

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

sseg            segemnt para stack ‘stack’
sseg            ends

zzzzzzseg       segment para public ‘zzzzzzseg’
zzzzzzseg       ends

CSEG            segment para public ‘CODE’
                assume  cs:cseg, ds:nothing

Int15Vect       dword   0
PSP             word    ?
; Adres portu dla typowego złącza joysticka:

JoyPort         equ     201h
JoyTrigger      equ     201h

CurrentReading  word    0

Pot             struct
PotMask         byte    0               ;maska potencjometru
DidCal          byte    0               ;czy potencjometr skalibrowany?
min             word    5000            ;minimalna wartość potencjometru
max             word    0               ;maksymalna wartość potencjometru
center          word    0               ;wartość potencjometru na środku
Pot             ends

Pot0            Pot     <1>
Pot1            Pot     <2>
Pot2            Pot     <8>

; SwapButtons-  0 jeśli powinniśmy użyć normalnego przycisku flightstick pro,
;               1 jeśli powinniśmy zamienić lewy i prawy przycisk

SwapButtons     byte    0

; SwBits-       wartość wejściowa czterech bitów z FlightStick Pro wybierając jeden z poniższych wzorców
;               dla danej  pozycji przełącznika

SwBits          byte    10h                             ;Sw4
```

```
                byte    0                               ;NA
                byte    0                               ;NA
                byte    0                               ;NA
                byte    40h                             ;Sw6
                byte    0                               ;NA
                byte    0                               ;NA
                byte    4                               ;Sw2

                byte    80h                             ;Sw7
                byte    0                               ;NA
                byte    0                               ;NA
                byte    8                               ;Sw3
                byte    20h                             ;Sw5
                byte    2                               ;Sw1
                byte    1                               ;Sw0
                byte    0                               ;NA

SwBitsL         byte    10h                             ;Sw4
                byte    0                               ;NA
                byte    0                               ;NA
                byte    0                               ;NA
                byte    40h                             ;Sw6
                byte    0                               ;NA
                byte    0                               ;NA
                byte    4                               ;Sw2

                byte    80h                             ;Sw7
                byte    0                               ;NA
                byte    0                               ;NA
                byte    2                               ;Sw3
                byte    20h                             ;Sw5
                byte    8                               ;Sw1

                byte    1                                       ;Sw0
                byte    0                                       ;NA

; Adres IDString przekazuje ponownie do funkcji wywołującej wywołanie testpresence. Cztery bajty przed
; IDString muszą zawierać liczbę porządkową i  bieżącą liczbę urządzenia

SerialNumber    byte    0, 0, 0
IDNumber        byte    0
IDString        byte    "CH Products: FlightStick Pro", 0
                byte    "Written by Randall Hyde" 0

;=============================================================================
;
; ReadPots-     AH zawiera bit maski określający , który potencjometr powinien być odczytany. Bit 0 jest
;               jedynką jeśli powinniśmy odczytać potencjometr 0, bit 1 jest jedynką jeśli mamy odczytać
;               potencjometr 1, bit 3 jest jedynką jeśli mamy odczytać potencjometr 3/ Wszystkie inne bity
;               będą zawierały zero.
;
;               Kod ten zwraca wartości potencjometrów w SI, BX, BP i DI dla potencjometrów 0,1,2 i 3
;
ReadPots        proc    near
                sub     bp, bp
                mov     si, bp
                mov     di, bp
                mov     bx, bp
```

```
;Oczekiwanie potencjometru na zakończenie ostatniej akcji:

                mov     dx, JoyPort
                out     dx, al.                 ;wyzwolenie potencjometru
                mov     cx, 400h
Wait4Pots;      in      al, dx
                and     al, 0Fh
                loopnz  Wait4Pots

;Okay, odczyt potencjometrów:

                mov     dx, JoyTrigger
                out     dx, al.                 ;wyzwolenie potencjometru
                mov     dx, JoyPort
                mov     cx, 8000h
PotReadLoop:    in      al, dx
                and     al, ah
                jz      PotReadDone
                shr     al, 1
                adc     si, 0
                shr     al, 1
                adc     bp, 0
                shr     al, 2
                adc     di, 0
                loop    PotReadLoop
PotReadDone:
                Ret
ReadPots        endp

;-----------------------------------------------------------------------------------------------------------------------
-
;
; Normalize-    BX zawiera wskaźnik do struktury potencjometru, AX zawiera wartość potencjometru.
;               Normalizujemy tą wartość zgodnie z kalibracją potencjometru.
;
; Notka:        DS musi wskazywać cseg zanim wywołamy ten podprogram

                assume  ds:cseg
Normalize       proc    near
                push    cx

; Sprawdzamy aby upewnić się, że proces kalibracji przeszedł spokojnie

                cmp     [bx].Pot.DidCal, 0
                je      BadNorm
                mov     dx, [bx].Pot.Center
                cmp     dx, [bx].Pot.Min
                jbe     BadNorm
                cmp     dx,[bx].Pot.Max
                jae     BadNorm

; Przycinamy tą wartość jeśli jest poza zakresem

                cmp     ax, [bx].Pot.Min
                ja      MinOkay
                mov     ax, [bx].Pot.Min

MinOkay:
                cmp     ax, [bx].Pot. Max
```

```
                jb      MaxOkay
                mov     ax, [bx].Pot.Max

MaxOkay:

; Skalujemy  centralnie:

                cmp     ax, [bx].Pot.Center
                jb      Lower128

;Skalujemy w zakresie 128..255:

                sub     ax, [bx].Pot.Center
                mov     dl, ah                          ;mnożenie przez 128
                mov     ah, al.
                mov     dh, 0
                mov     al, dh
                shr     dl, 1
                rcr     ax, 1
                mov     cx, [bx].Pot.Max
                sub     cx, [bx].Pot.Center
                jz      BadNorm                 ;zapobiegamy dzieleniu przez zero
                div     cx                      ;obliczamy wartość znormalizowaną
                add     ax, 128                 ;skalujemy zakres 128.255
                cmp     ah, 0
                je      NormDone
                mov     ax, 0ffh                ;wynik musi się zmieścić w 8 bitach
                jmp     NormDone

;Skalujemy w zakresie 0..127:

Lower128:       sub     ax,[bx].Pot.Min
                mov     dl, ah                  ;mnożenie przez 128
                mov     ah, al.
                mov     dh, 0
                mov     al, dh
                shr     dl, 1
                rcr     ax, 1
                mov     cx, [bx].Pot.Center
                sub     cx, [bx].Pot.Min
                jz      BadNorm
                div     cx                      ;obliczenie wartości normalizowanej
                cmp     ax, 0
                je      NormDone
                mov     ax, 0ffh                ;wynik musi się zmienić  w 8 bitach!
                jmp     NormDone

BadNorm:        sub     ax, ax
NormDone:       pop     cx
                Ret
Normalize       endp
                asume   ds:nothing
;=============================================================================
; Funkcja obsługi INT 15h
;=============================================================================
; Chociaż jest to zdefiniowane jako procedury bliskie, nie są w rzeczywistości procedurami. Kod MyInt15
; skacze do każdej z nich z BX, daleki adres powrotu i flag usytuowanych na stosie. Każdy  z tych
; podprogramów musi właściwie obsłużyć stos.
;
```

```
;-------------------------------------------------------------------------------------------------------------------------
-
; BIOS-         obsługuje dwie funkcje BIOS, DL=0 odczytującą przełączniki, DL=1 odczytującą
;               potencjometry. Dla podprogramów BIOS zignorujemy cooley switch (the hat) i po prostu
;               odczytamy pozostałe cztery przełączniki.

BIOS            proc    near
                cmp     dl, 1                   ;zobacz czy podprogram przełącznika czy
                jb      Read4Sw                 ;potencjometru
                je      ReadBIOSPots
                pop     bx
                jmp     cs:Int15Vect

Read4sw:        push    dx
                mov     dx, JoyPort
                in      al, dx
                shr     al, 4
                mov     bl, al
                mov     bh, 0
                cmp     cs: SwapButtons, 0
                je      DoLeft2
                mov     al, cs:SwBitsL[bx]
                jmp     SBDone

DoLeft2;        mov     al, cs:SwBits[bx]
SBDone:         rol     al, 4               ;odłożenie Sw0..3 w górnych bitach I czynimy 0=w dole
                not     al.                 ;podobnie jak złącze gier
                pop     dx
                pop     bx
                iret

ReadBIOSPots:   pop     bx                  ;zwraca wartość w BX
                push    si
                push    di
                push    bp
                mov     ah, 0bh
                call    ReadPots
                mov     ax, si
                mov     bx, bp
                mov     dx, di
                sub     cx, cx
                pop     bp
                pop     di
                pop     si
                iret
BIOS            endp

;-------------------------------------------------------------------------------------------------------------------------
-
;
; ReadPot-      Na wejściu, DL zawiera numer potencjometru do odczytu. Odczytujemy i normalizujemy
;               potencjometr i zwracamy wynik w AL.

                assume  ds:cseg
ReadPot         proc    near
;;;;;;;;;;;;;;  push    bx                  ;już na stosie
                push    ds.
                push    cx
                push    dx
```

```
                push    si
                push    di
                push    bp

                mov     bx, cseg
                mov     ds., bx

                cmp     dl, 0
                jne     Try1
                mov     ah, Pot0.PotMask
                call    ReadPots
                lea     bx, Pot0
                mov     ax, si
                call    Normalize
                jmp     GotPot

Try1:           cmp     dl, 1
                jne     Try3
                mov     ah, Pot1.PotMask
                call    ReadPots
                lea     bx, Pot1
                mov     ax, bp
                call    Normalize
                jmp     GotPot
Try3:           cmp     dl, 3
                jne     BadPot
                mov     ah, Pot3.PotMask
                call    ReadPots
                lea     bx, Pot3
                mov     ax, di
                call    Normalize
                jmp     GotPot

BadPot:         sub     ax, ax                  ;Pytanie: czy powinniśmy to przekazać
                                                ;czy zwrócić zero
GotPot:         pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     ds
                pop     bx
                iret
ReadPot         endp
                assume  ds.:nothing

;-----------------------------------------------------------------------------------------------------------------------
-
;
; ReadRaw-      Na wejściu DL zawiera numer potencjometru od odczytu. Odczytuje ten potencjometr i zwraca
;               nieznormalizowany wynik w AL.

                assume  ds.:cseg
ReadRaw         proc    near
;;;;;;;;;;;;;;  push    bx                      ;już na stosie
                push    ds.
                push    cx
                push    dx
                push    si
```

```
                push    di
                push    bp

                mov     bx. Cseg
                mov     ds., bx
                cmp     dl, 0
                jne     Try1
                mov     ah, Pot0.PotMask
                call    ReadPots
                mov     ax, si
                jmp     GotPot

Try1:           cmp     dl, 1
                jne     Try3
                mov     ah, Pot1.PotMask
                call    ReadPots
                mov     ax, bp
                jmp     GotPot
Try3:           cmp     dl, 3
                jne     BadPot
                mov     ah, Pot3.Mask
                call    ReadPots
                mov     ax, di
                jmp     GotPot

BadPot:         sub     ax, ax                          ;zwracane zero
GotPot:         pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     ds.
                pop     bx
                iret
ReadRaw         endp
                assume  ds.:nothing

;-------------------------------------------------------------------------------------------------------------------------------
-
; Read4Pots-    Odczytuje potencjometry zero, jeden ,dwa I trzy zwracając ich wartości w AL, AH, DL I DH.
;               Ponieważ Flighstick Pro nie ma zainstalowanego potencjometr u dwa zwraca zero tej sytuacji.

Read4Pots       proc    near
;;;;;;;;;;;;;;;;;       push    bx                      ;już na stosie
                push    ds.
                push    cx
                push    si
                push    di
                push    bp

                mov     dx, cseg
                mov     ds., dx
                mov     ah, 0bh                         ;odczyt potencjometru 0, 1 i 3
                call    ReadPots

                mov     ax, si
                lea     bx, Pot0
                call    Normalize
                mov     cl, al.
```

```
                mov     ax, bp
                lea     bx, Pot1
                call    Normalize
                mov     ch, al
                mov     ax, di
                lea     bx, Pot3
                call    Normalize
                mov     dh, al                          ;wartość potencjometru 3
                mov     ax, cx                          ;potencjometry 0 I 1
                mov     dl, 0                           ;potencjometr 2 nie istnieje

                pop     bp
                pop     di
                pop     si
                pop     cx
                pop     ds.
                pop     bx
                iret
Read4Pots       endp

;-----------------------------------------------------------------------------------------------------------------------
-
; CalPot-       Kalibrujemy potencjometr określony przez DL. Na wejściu AL. Zawiera minimalną wartość (
;               niemniej niż 256!), BX zawiera wartość maksymalną, a CX zawiera wartość centralną.

                assume  ds.:cseg
CalPot          proc    near
                pop     bx                              ;odzyskanie wartości maksymalnej
                push    ds.
                push    si
                mov     si, cseg
                mov     ds., si

; Sprawdzanie parametrów, sortowanie według rosnącego porządku:

                mov     ah, 0
                cmp     bx, cx
                ja      GoodMax
                xchg    bx, cx
GoodMax:        cmp     ax, cx
                jb      GoodMin
                xchg    ax, cx
GoodMin:        cmp     cx, bx
                jb      GoodCenter
                xchg    cx, bx
GoodCenter:

;Okay, wymyślimy , kto wspiera kalibrację

                lea     si, Pot0
                cmp     dl , 1
                jb      DoCal
                lea     si, Pot1
                je      DoCal
                cmp     dl, 3
                jne     CalDone
                lea     si, Pot3

DoCal:          mov     [si].Pot.min, ax
```

```
                mov     [si].Pot.max, bx
                mov     [si].Pot.center, cx
                mov     [si].PotDidCal, 1
CalDone:        pop     si
                pop     ds
                iret
CalPot          endp
                assume  ds:nothing

;-----------------------------------------------------------------------------------------------------------------------
-
; TestCal-      sprawdzenie czy potencjometr określony w DL nie została już skalibrowany
;
                assume  ds.:cseg
TestCal         proc    near
;;;;;;;;;;;;;   push    bx                              ;już na stosie
                push    ds.
                mov     bx, cseg
                mov     ds., bx

                sub     ax, ax                          ;zakładamy brak kalibracji
                lea     bx, Pot0
                cmp     dl, 1
                jb      GetCal
                lea     bx, Pot1
                je      GetCal
                cmp     dl, 3
                jne     BadCal
                lea     bx, Pot3

GetCal:         mov     al., [bx].Pot.DidCal
                mov     ah, 0
BadCal:         pop     ds
                pop     bx
                iret
TestCal         endp
                assume  ds:nothing

;-----------------------------------------------------------------------------------------------------------------------
-
;
; ReadSw-       odczyt przełącznika którego numer pojawia się w DL

SwTable         byte    11100000b, 11010000b, 01110000b, 10110000b
                byte    00000000b, 11000000b, 01000000b, 10000000b

SwTableL        byte    11100000b, 10110000b, 01110000b, 11010000b
                byte    00000000b, 11000000b, 01000000b, 10000000b

ReadSw          proc    near
;;;;;;;;;;;;   push    bx                                      ;już na stosie
                mov     bl, dl                                  ;zachowanie przełącznika do odczytu
                mov     bh, 0
                mov     dx, JoyPort
                in      al., dx
                and     al., 0f0h
                cmp     cs;SwapButtons, 0
                je      DoLeft0
                cmp     al, cs:SwTableL[bx]
```

```
                jne     NotDown
                jmp     IsDown

DoLeft0:        cmp     al, cs:SwTable[bx]
                jne     NotDown
IsDown:         mov     ax, 1
                pop     bx
                iret
NotDown:        sub     ax, ax
                pop     bx
                iret
ReadSw          endp

;-------------------------------------------------------------------------------------------------------------------------
-;
; Read16Sw-     Odczyt wszystkich ośmiu przełączników I zwrot ich wartości w AX

Read16Sw        proc    near
;;;;;;;;;;;;;;  push    bx                                      ;już na stosie
                mov     ah, 0
                mov     dx, JoyPort
                in      al., dx
                shr     al., 4
                mov     bl, al.
                mov     bh, 0
                cmp     cs:SwapButtons, 0
                je      DoLeft1
                mov     al, cs:SwBitsL[bx]
                jmp     R8Done

DoLeft1:        mov     al, cs:SwBits[bx]
R8Done:         pop     bx
                iret
Read16Sw        endp

;**************************************************************************************
*
;
; MyInt15-      Aktualizacja podprogramu BIOS INT 15 sterującego odczytem joysticka
;
MyInt 15        proc    far
                push    bx
                cmp     ah, 84h                                 ;kod joysticka?
                je      DoJoystick
OyherInt15:     pop     bx
                jmp     cs:Int15Vect
DoJoystick:     mov     bh, 0
                mov     bl, dh
                cmp     bl, 80h
                jae     VendorCalls
                cmp     bx, JmpSize
                jae     OtherInt15
                shl     bx, 1
                jmp     wp  cs:jmptable[bx]
jmptable        word    BIOS
                word    ReadPot, Read4Pots, CalPot, TestCal
                word    ReadRaw, OtherInt15, OtherInt15
                word    ReadSw, Read16Sw
JmpSize         =       ($-jmptable)/2
```

```
;Obsługa określonej funkcji

VendorCalls:    je      RemoverDriver
                cmp     bl, 81h
                je      TestPresence
                pop     bx
                jmp     cs: Int15Vect

;TestPresence-  zwraca zero w AX I wskazuje na ID ciągu w ES:BX

TestPresence:   pop     bx                      ;pobranie starej wartości ze stosu
                sub     ax, ax
                mov     bx, cseg
                mov     es, bx
                lea     bx, IDString
                iret

;RemoverDriver- Jeśli nie ma żadnych sterowników załadowanych po tym w pamięci, rozłączamy go i usuwamy
;               z pamięci.

RemoverDriver:
                push    ds.
                push    es
                push    ax
                push    dx

                mov     dx, cseg
                mov     ds., dx

;zobaczmy czy ostatni podprogram zaktualizował INT 15h

                mov     ax, 3515h
                int     21h
                cmp     bx offset MyInt15h
                jne     CantRemove
                mov     bx, es
                cmp     bx, wp seg MyInt15
                jne     CantRemove

                mov     ax, PSP                 ;zwolnienie pamięci
                mov     es, ax
                push    es
                mov     ax, es:[2ch]            ;najpierw zwalniamy blok środowiska
                mov     es, ax
                mov     ah, 49h
                int     21h

                pop     es                      ;teraz zwalniamy przestrzeń programu
                mov     ah, 49h
                int     21h

                lds     dx, Int15Vect           ;przywrócenie poprzedniego wektora int
                mov     ax, 2515h
                int     21h

CantRemove:     pop     dx
                pop     ax
                pop     es
```

```
                pop     ds.
                pop     bx
                iret
MyInt15         endp
cseg            ends

;Poniższy segment jest odrzucany jeśli  ten kod staje się rezydentny

Initialize      segment para public     'INIT'
                assume  cs: Initialize, ds:cseg
Main            proc
                mov     ax, cseg                        ;pobranie wskaźnika do segmentu zmiennych
                mov     es, ax
                mov     es: PSP, ds.                    ;zachowanie PSP
                mov     ds., ax

                mov     ax, zzzzzzseg
                mov     es, ax
                mov     cx, 100h
                meminit2

                print
                byte    „Standard Game Device Interface driver”, cr, lf
                byte    “CH Products Flightstick Pro”, cr,lf
                byte    “Written by Randal Hyde”, cr, lf
                byte    cr, lf
                byte    “ ‘FSPSGDI LEFT’ zamienia lewy i prawy przycisk dla „
                byte    „lewo ręcznych graczy”, cr, lf
                byte    „ ‘FSPGDI REMOVE’ usuwa sterownik z pamięci”
                byte    cr, lf, lf
                byte    0

                mov     ax, 1
                argv                                    ;jeśli nie ma parametrów, pusty ciąg
                stricmpl
                byte    „LEFT”, 0
                jne     NoLEFT
                mov     SwapButtons, 1
                print
                byte    „Lewy i prawy przycisk zmienione”,cr,lf,0
                jmp     SwappedLeft

NoLEFT:         stricmpl
                byte    „REMOVE”, 0
                jne     NoRmv
                mov     dh, 81h
                mov     ax, 84ffh
                int     15h                             ;sprawdzamy czy już nie załadowaliśmy
                test    ax, ax
                jz      Installed
                print
                byte    “Sterownik SGDI nie jest obecny w pamięci, polecenie REMOVE „
                byte    „zignorowane.”, cr, lf, 0
                mov     ax, 4c01h                       ;wyjście do DOS’a
                int     21h

Installed:      mov     ax, 8400h
                mov     dh, 80h                         ;funkcja usuwania
                int     15h
```

```
                mov     ax, 8400h
                mov     dh, 81h                         ;funkcja TestPresence
                int     15h
                cmp     ax, 0
                je      NotRemoved
                print
                byte    "Usunięcie  sterownika SGDI  z pamięci zakończone pomyślnie.", cr, lf,0
                mov     ax, 4c01h                       ;wyjście do DOS'a
                int     21h

NotRemoved:     print
                byte    „Sterownik SGDI jest jeszcze obecny w pamięci." ,cr, lf, 0
                mov     ax, 4c01h                       ;wyjście do DOS'a
                int     21h

NoRmv:

;Okay, aktualizujemy INT 15 i pozostawiamy w pamięci

SwappedLeft:    mov     ax, 3515h
                int     21h
                mov     wp  Int15Vect, bx
                mov     wp Int15Vect+2, es

                mov     dx, cseg
                mov     ds, dx
                mov     dx, offset MyInt15
                mov     ax, 2515h
                int     21h

                mov     dx, cseg
                mov     ds, dx
                mov     dx, seg Initialize
                sub     dx, ds:psp
                add     dx, 2
                mov     ax, 3100h
                int     21h
Main            endp

Initialize      ends

sseg            segment para  stack ' stack'
                word    128  dup (0)
endstk          word    ?
sseg            ends

zzzzzzseg       segment para  public  'zzzzzzseg'
                byte    16  dup (0)
zzzzzzseg       ends
                end     Main
```

24.7 POPRAWIANIE ISTNIEJĄCYCH GIER

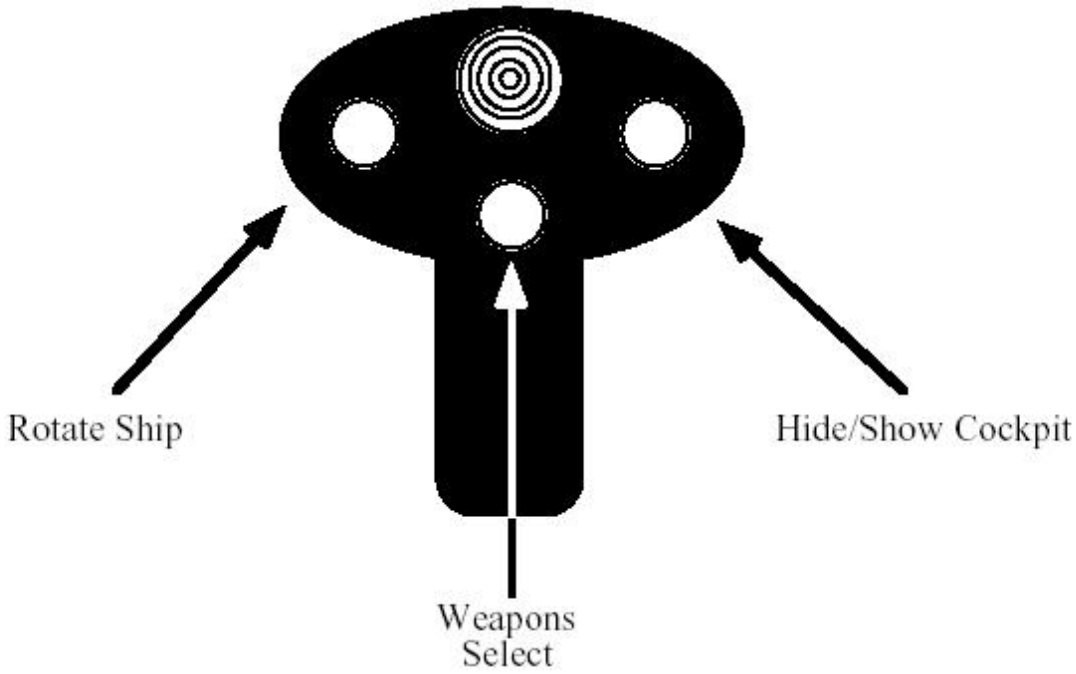
Być może nie jesteś gotowy do napisania gry za milion dolarów. Być może chciałbyś uczynić bardziej przyjemniejszą grę którą już posiadasz . Cóż ,sekcja ta dostarczy praktycznej aplikacji z półrezydentnym programem , który aktualizuje grę Lucas Arts Xwing (symulacja Star Wars). Program aktualizuje grę Xwing wykorzystując specjalne cechy CH Products' FlightStick Pro. W szczególności pozwala zastosować

potencjometr dławiący w FSP do sterowania szybkością pojazdów kosmicznych. Pozwala również oprogramować każdy z przycisków do czterech ciągów ośmioznakowych każdy.

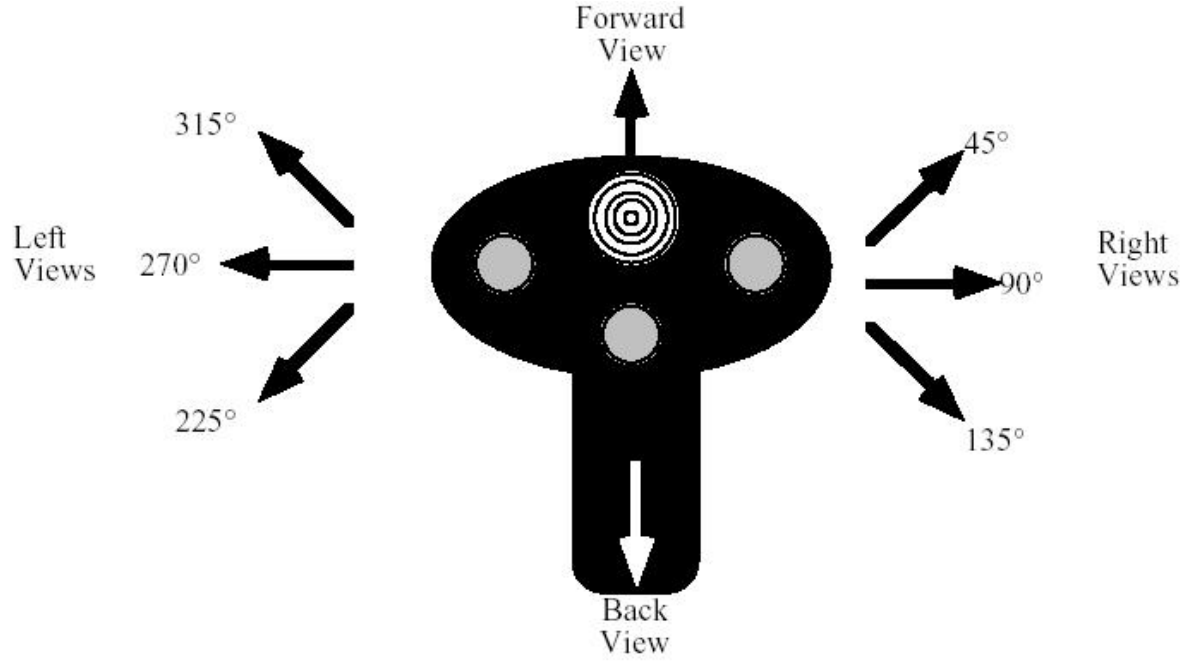
Opisując jak zaktualizować istniejącą grę, opiszemy krótko jak ta aktualizacja się rozwinęła. Łatka FSPXW została wywołana przez debugger Soft-ICE. Program ten pozwala nam ustawiać punkty przerwań gdziekolwiek w 80386 lub późniejszych procesorach w określonym porcie I/O. Ustawiając punkt przerwania pod adresem I/O 201h podczas działania xwing.exe, zatrzymujemy program Xwing, kiedy zdecydujemy się odczytać wejścia analogowe i przełączniki. Dizasemblując kod otaczający tworzący kompletny joystick a przycisk odczytuje podprogramy. PO zlokalizowaniu tego podprogramu, było łatwo dość napisać program do przeszukiwania pamięci dla kodu i aktualizacji w skokach kodu w programie FSPXW

Zauważmy ,że oryginalny kod joysticka wewnątrz Xwing działa doskonale z FSP Jedynym powodem aktualizacji kodu joysticka jest to ,że nasz kod może odczytać dławik i podjąć stosowną akcję.

Podprogram przycisku to zupełnie inna historia. Łatka FSPXW musi pobrać sterowanie z podprogramu przycisku Xwing ponieważ użytkownik FSPXW może chcieć przedefiniować przycisk rozpoznany przez Xwing dla innego celu. Dlatego też, ilekroć XWing wywołuje podprogram przycisku, sterowanie jest przekazywane do podprogramu przycisku wewnątrz FSPXW, który decyduje czy przekazać rzeczywistą informację z powrotem do Xwing lub udawać przycisk w górnej pozycji ponieważ przyciski te są predefiniowane dla innych funkcji. Domyślnie (chyba, że zmienimy kod źródłowy) przyciski są następująco oprogramowane:



Oprogramowanie **cooley switcha** demonstruje interesującą cechę łatki FSPXW: możemy oprogramować do czterech różnych ciągów na każdym przycisku. Pierwszy raz naciskając przycisk, FSPXW emituje pierwszy ciąg, za drugim razem naciskając przycisk emituje drugi ciąg, potem trzeci a w końcu czwarty. Jeśli ciąg jest pusty, FSPXW przeskakuje ciąg. Łatka FSPXW używa **cooley switcha** do wyboru widoku z kokpitu. Naciskając **cooley switch** do przodu, wyświetla się widok z przodu, cofając **cooley switch** do tyłu mamy widok z tyłu. Jednakże gra Xwing dostarcza trzech lewych i prawych widoków. Naciskając **cooley switch** w lewo lub prawo po raz pierwszy mamy widok pod kątem 45 stopni. Naciskając po raz drugi mamy widok pod katem 90 stopni. Naciskając w lewo lub w prawo po raz trzeci mamy widok pod kątem 135 stopni. Poniższy diagram pokazuje domyślne oprogramowanie w **cooley switch**:



Słowo ostrzeżenia dotyczące tej łatki: działa tylko z podstawową wersją gry Xwing. Nie wspiera modułów ponad standardowych (Imperial Pursuit, B-Wing, Tie Fighter, itp.) . Co więcej, łatka ta zakłada, że podstawowy kod Xwing nie zmienił się od lat. Może być tak, że nowsze wersje gry Xwing używają nowszych Podprogramów joysticka i k0d powiązany z tą aplikacją może nie być zdolny do rozpoznania i zaktualizowania tego nowego podprogramu. Łatka ta będzie wykrywała taką sytuację i nie zaktualizuje Xwing jeśli wystąpi taki przypadek Musimy mieć wystarczającą ilość wolnej pamięci RAM dla tej łatki, Xwing i wszystko inne załadowane do pamięci w tym samym czasie (dokładną ilość RAM potrzebną Xwing zależy od zainstalowanych cech, pełna instalacja systemu wymaga nie mniej niż 610 K).

Bez dalszych wstępów, mamy tu kod FSPXW:

```
                .286
                page    58, 132
                name    FSPXW
                title   FSPXW (Flightstick Pro driver dla XWING)
                subttl  Copyright  (C )  1994 Randall Hyde

; FSPXW.EXE
;
;       Usage:
;               FSPXW
;
; Program ten wykonuje program XWING.EXE i aktualizuje go do zastosowania z Flightstick Pro

byp             textequ  < byte ptr >
wp              textequ  < word ptr >

cseg            segment para publick 'CODE'
cseg            ends

sseg            segment para stack 'STACK'
sseg            ends

zzzzzzseg       segment para  public 'zzzzzzseg'
zzzzzzseg       ends
```

```
                include         stdlib.a
                includelib      stdlib.lib
                matchfuncs

                ifndef          debug
Installation    segment para public 'Install'
Instalation     ends
                endif

CSEG            segment para public 'CODE'
                assume  cs:cseg, ds:nothing

; Wektor przerwania zegarowego

Int1Cvect       dword   ?

;PSP-    Prefiks Segmentu Programu. Musimy zwolnić pamięć zanim uruchomimy rzeczywisty program

PSP             word    0

;Program ładujący strukturę danych (dla DOS)

ExecStruct      word    0                       ;uzywamy rodzicielskiego bloku środowiska
                dword   CmdLine                 ;dla parametrów linii poleceń
                dword   DfltFCB
                dword   DfltFCB
LoadSSSP        dword   ?
LoadCSIP        dword   ?
PgmName         dword   Pgm

;Zmienne dla potencjometru dławiącego.
; LastThrottle zawiera ostatnio wysłany znak (więc wysyłamy tylko jedną kopię)
;ThrtlCntDn zlicza liczbę razy wywołania podprogramu dławienia.

LastThrottle    byte    0
ThrtlCntDn      byte    10

;ButtonMask-     używamy do maskowania oprogramowanych przycisków kiedy gra odczytuje rzeczywiste
;               przyciski

ButtonMask      byte    0f0h

;Poniższe zmienne pozwalają użytkownikowi przeprogramować przyciski

KeyRdf          struct
Ptrs            word    ?                       ;pole PTRx wskazuje na cztery możliwe ciągi z 8
ptr2            word    ?                       ;znakami każdy. Każdy przycisk naciskany jest
ptr3            word    ?                       ;cyklicznie przez te ciągi.
ptr4            word    ?
Index           word    ?                       ;Indeks do kolejnego ciągu wyjściowego
Cnt             word    ?
Pgmd            word    ?                       ;Flag = 0 jeśli nie predefiniowane
KeyRdf          ends

; Kody  Left są wyjściowe jeśli cooley switch  jest naciśnięty w lewo. Zauważmy, że ciagi są w
; w rzeczywistości ciągami słów zakończonymi zerami.

Left            KeyRdf <Left1, Left2, Left3, Left4, 0, 6, 1>
```

```
Left1           word    ‘7’, 0
Left2           word    ‘4’,0
Left3           word    ‘1’, 0
Left4           word    0

; Kody Right są wyjściowe jeśli cooley switch jest przesunięty w prawo

Right           KeyRdf <Right1, Right2, Right3, Right4, 0, 6 ,1>
Right1          word    ‘9’, 0
Right2          word    ‘6’, 0
Right3          word    ‘3’, 0
Right4          word    0

; Kody Up są wyjściowe jeśli cooley switch jest przesunięty w górę

Up              KeyRdf <Up1, Up2, Up3, Up4, 0, 2, 1>
Up1             word    ‘8’, 0
Up2             word    0
Up3             word    0
Up4             word    0

;Kody DownKeys są wyjściowe jeśli cooley switch  jest przesunięty w dół

Down            KeyRdf <Down1, Down2, Down3, Down4, 0, 2, 1>
Down1           word    ‘2’, 0
Down2           word    0
Down3           word    0
Down3           word    0

; Kody Sw0 są wyjściowe jeśli użytkownik popchnie wyzwalanie.(Ten przełącznik nie jest redefiniowany
          )

Sw0             KeyRdf <Sw01, Sw02, Sw03, Sw04, 0 ,0 ,0 >
Sw01            word    0
Sw02            word    0
Sw03            word    0
Sw04            word    0

; Kody Sw1 są wyjściowe jeśli użytkownik wciśnie Sw1 (lewy przycisk jeśli  użytkownik nie zmienił
; lewego i prawego przycisku). Nie redefiniowany

Sw1             KeyRdf <Sw11, Sw12, Sw13, Sw14, 0 ,0 ,0>
Sw11            word    0
Sw12            word    0
Sw13            word    0
Sw14            word    0

;Kody Sw2 są wyjściowe jeśli użytkownik naciśnie Sw2 (środkowy przycisk)

Sw2             KeyRdf <Sw21, Sw22, Sw23, Sw24, 0, 2, 1>
Sw21            word    ‘w’, 0
Sw22            word    0
Sw23            word    0
Sw24            word    0

;Kody Sw3 są wyjściowe jeśli użytkownik naciśnie Sw3 (prawy przycisk jeśli użytkownik nie zmienił lewego
; i prawego przycisku)

Sw3             KeyRdf <Sw31,Sw32, Sw33, Sw34, 0 ,0 ,0>
```

```
Sw31            word    0
Sw32            word    0
Sw33            word    0
Sw34            word    0

;Przycisk stanu przełącznika:

CurSw           byte    0
LastSw          byte    0

;*****************************************************************************
*; Łatka FSPXW zaczyna się tutaj. Jest to część rezydentna w pamięci Tylko należy włożyć kod, który ma być
; obecny w czasie uruchamiania lub musi pozostać rezydentny po zwolnieniu pamięci.
;*****************************************************************************
*

Main            proc
                mov     cs:PSP, ds
                mov     ax, cseg                        ;pobranie wskaźnika do segmentu zmiennych
                mov     ds., ax

;Pobranie aktualnego wektora przerwania INT 1Ch

                mov     ax, 351ch
                int     21h
                mov     wp Int1Cvect, bx
                mov     wp  Int!cVect+2, es

:Poniższe wywołanie MEMINIT zakłada ,że nie występują błędy.

                Mov     ax, zzzzzzseg
                mov     es, ax
                mov     cx, 1024/16
                meminit2

; Wykonujemy inicjalizację przed uruchomieniem gry. To są funkcje kodu inicjalizującego pobierającego kopię
; przed rzeczywistym uruchomieniem XWING

                call    far ptr ChkBIOS15
                call    far ptr Identify
                call    far ptr Calibrate

; Jeśli jakiś przełącznik został oprogramowany, usuwamy ten przełącznik z ButtonMask

                mov     al., 0f0h                       ;zakładamy ,że wszystkie przyciski są OK
                cmp     sw0.pgmd, 0
                je      Sw0NotPgmd
                and     al., 0e0h                       ;usuwamy sw0 ze współzawodnictwa
Sw0NotPgmd:

                cmp     sw1.pgmd, 0
                je      Sw1NotPgmd
                and     al., 0d0h                               ;usuwamy sw1

Sw1NotPgmd:

                cmp     sw2.pgdm, 0
                je      Sw2NotPgmd
                and     al, 0b0h                                ;usuwamy Sw2
```

```
Sw2NotPgmd:

                cmp     sw3.pgdm, 0
                je      Sw3NotPgmd
                and     al, 070h                        ;usuwmay sw3

Sw3NotPgmd:

                mov     ButtonMask, al          ;zachowujemy wynik jako maskę przycisku

; Teraz, zwalniamy pamięć z ZZZZZZSEG robiąc miejsce dla XWING. Notka: Absolutnie nie wywołujemy
; podprogramów Biblioteki Standardowej UCR w tym miejscu! (ExitPgm jest OK., jest to makro które
; wywołuje DOS). Zauważmy ,ze  po wykonaniu tego kodu, żaden kod i dane z zzzzzzseg są poprawne.

                mov     bx, zzzzzzseg
                sub     bx, PSP
                inc     bx
                mov     es, PSP
                mov     ah, 4ah
                int     21h
                jnc     GoodRealloc
                print
                byte    "Memory allocation error"
                byte    c, lf,0
                jmp     Quit

GoodRealloc:

;Teraz ładujemy program XWING do pamięci:

                mov     bx, seg ExecStruct
                mov     es, bx
                mov     bx, offset ExecStruct                   ;wskaźnik do rekordu programu
                lds     dx, PgmName
                mov     ax, 4b01h                               ;ładowanie pgm
                int     21h
                jc      Quit                                    ;jeśli błędnie załadowano plik

;Poszukiwanie kodu joysticka w pamięci:

                mov     si, zzzzzzseg
                mov     ds., si
                xor     si, si

                mov     di, cs
                mov     es, di
                mov     di, offset JoyStickCode
                mov     cx, joyLength
                call    FindCode
                jc      Quit                                    ;jeśli nie znaleziono kodu joysticka

;Łatanie kodu joysticka XWING:

                mov     byp  ds.:[si], 09ah                     ;dalekie wywołanie
                mov     wp  ds.:[si+1], offset ReadGame
                mov     wp  ds:[si+3], cs

;Znajdoanie kodu Button
```

```
                mov     si, zzzzzzseg
                mov     ds, si
                xor     si, si

                mov     di, cs
                mov     es, di
                mov     di, offset ReadSwCode
                mov     cx, ButtonLength
                call    FindCode
                jc      Quit

;Łatanie kodu przycisku

                mov     byp  ds:[si], 9ah
                mov     wp  ds:[si+1], offset ReadButtons
                mov     wp  ds:[si+3], cs
                mov     byp ds:[si+5], 90h                              ;NOP

: Aktualizacja naszego programu obsługi przerwania zegarowego:

                mov     ax, 251ch
                mov     dx, seg MyInt1C
                mov     ds, dx
                mov     dx, offset MyInt1C
                int     21h

;Okay, zaczynamy uruchamianie programu XWING.EXE

                mov     ah, 62h                         ;pobranie PSP
                int     21h
                mov     ds., bx
                mov     es, bx
                mov     wp  ds.:[10], offset Quit
                mov     wp ds:[12], cs
                mov     ss, wp  cseg:LoadSSSP+2
                mov     sp, wp cseg:LoadCSIP
                jmp     dword  ptr cseg:LoadCSIP

Quit:           lds     dx, cs:Int1Cvect                ;przywrócenie wektora zegarowego
                mov     ax, 251ch
                int     21h
                ExitPgm
Main            endp

;*****************************************************************************************
*
;
; ReadGame-     Podprogram wywoływany kiedy Xwing odczytuje joystick. Przy co dziesiątym wywołaniu
;               będzie odczytywał potencjometru dławiącego i wysyła właściwy znak do bufora
;               roboczego, jeśli to konieczne

                assume  ds.:nothing
ReadGame        proc    far
                dec     cs: ThrtlCntDn                  ;robi to tylko co 10 raz
                jne     SkipThrottle                    ;XWING wywołuje podprogram joysticka
                mov     cs: ThrtlCntDn, 10

                push    ax
                push    bx                              ;nie musimy zachowywać bp, dx lub cx
```

```
                push    di                              ;ponieważ XWING je zachowuje

                mov     ah,84h
                mov     dx, 103h                        ;odczyt potencjometru dławiącego
                int     15h

;Konwersja wartości zwracanych przez podprogram potencjometru do czterech znaków 0..63:"\", 64..127:"[„ ,
; 128..191: „]", 192..255:<bs>, oznaczając odpowiednio zero, 1/3, 2/3  i pełna moc

                mov     dl, al.
                mov     ax, "\"                         ;zero mocy
                cmp     dl, 192
                jae     SetPower
                mov     ax, „[„                         ; 1/3 mocy
                cmp     dl, 128
                jae     SetPower
                mov     ax, „]"                         ;2/3 mocy
                cmp     dl, 64
                jae     SetPower
                mov     ax, 8                           ;BS, pełna moc
SetPower:       cmp     al., cs:LastThrottle
                je      SkipPIB
                mov     csL:LastThrottle, al
                call    PutIntoBuffer

SkipPIB         pop     di
                pop     bx
                pop     ax
SkipThrottle:   neg     bx                              ;XWING zwraca dane w tych rejestrach.
                neg     di                              ;aktualizujemy instrukcje NEG i STI
                sti
                ret
ReadGame        endp

                assume ds: nothing
RaedButtons     proc    far
                mov     ah, 84h
                mov     dx, 0
                int     15h
                not     al
                and     al, ButtonMask                  ;wyłączamy pgmd przycisków
                ret
ReadButtons     endp

; MyInt1C-      wywoływana co 1/18 sekundy. Odczytuje przełączniki i decyduje czy powinna  zostawić
jakieś
;               znaki w buforze roboczym

                assume  ds.:cseg
MyInt1C         proc    far
                push    ds
                push    ax
                push    bx
                push    dx
                mov     ax, cseg
                mov     ds, ax
                mov     al, CurSw
                mov     LastSw, al
```

```
                mov     dx, 900h                ;odczyt 8 przełączników
                mov     ah, 84h
                int     15h

                mov     CurSw, al.
                xor     al, LastSw              ;zobacz czy są zmiany
                jz      NoChanges
                and     al., CurSw              ;zobacz czy sw zszedł w dół
                jz      NoChanges
```

; Jeśli przełącznik był zszedł w dół ,wyjście stosownie ustawi kody klawiszy dla niego, jeśli ten klawisz jest ;aktywny. Zauważmy ,ze naciśnięcie * jakiegoś* klawisza zresetuje wszystkie inne indeksy klawiszy

```
                test    al., 1                  ;zobacz czy Sw0 (wyzwalacz) pchnięto
                jz      NoSw0
                cmp     Sw0.Pgmd, 0
                je      NoChanges
                mov     ax, 0
                mov     Left.Index, ax          ;reset indeksów klawiszy dla wszystkich klawiszy
                mov     Right.Index , ax        ;z wyjątkiem SW0
                mov     Up.Index, ax
                mov     Down.Index, ax
                mov     Sw1.Index, ax
                mov     Sw2.Index, ax
                mov     Sw3.Index, ax
                mov     bx, Sw0.Index
                mov     ax, Sw0.Index
                mov     bx, Sw0.Ptrs[bx]
                add     ax, 2
                cmp     ax, Sw0.Cnt
                jb      SetSw0
                mov     ax, 0
SetSw0:         mov     Sw0.Index, ax
                call    PutStrInBuf
                jmp     NoChanges

NoSw0:          test    al, 2                   ;sprawdzamy czy naciśnięto SW1
                jz      NoSw1
                cmp     Sw1.Pgmd, 0
                je      NoChanges
                mov     ax, 0
                mov     Left.Index, ax          ;reset indeksów klawiszy dla wszystkich
                mov     Right.Index, ax         ;klawiszy z wyjątkiem Sw1
                mov     Up.Index, ax
                mov     Down.Index, ax
                mov     Sw0.Index, ax
                mov     Sw2.Index, ax
                mov     Sw3.index, ax
                mov     bx, Sw1.Index
                mov     ax, Sw1.Index
                mov     bx, Sw1.Ptrs[bx]
                add     ax, 2
                cmp     ax, Sw1.Cnt
                jb      SetSw1
                mov     ax, 0
SetSw1:         mov     Sw1.Index, ax
                call    PutStrInBuf
                jmp     NoChanges
```

```
NoSw1:  test    al, 4                   ;zobacz czy Sw2 (środkowy sw jest naciśniety)
        jz      NoSw2
        cmp     Sw2.Pgmd, 0
        je      NoChanges
        mov     ax, 0
        mov     Left.Index, ax          ;reset indeksów klawiszy dla wszystkich
        mov     Right.Index, ax         ;klawiszy z wyjątkiem Sw2
        mov     Up.Index, ax
        mov     Down.Index, ax
        mov     Sw0.Index, ax
        mov     Sw2.Index, ax
        mov     Sw3.index, ax
        mov     bx, Sw2.Index
        mov     ax, Sw2.Index
        mov     bx, Sw2.Ptrs[bx]
        add     ax, 2
        cmp     ax, Sw2.Cnt
        jb      SetSw2
        mov     ax, 0
SetSw2: mov     Sw2.Index, ax
        call    PutStrInBuf
        jmp     NoChanges
NoSw2:  test    al, 8h                  ;zobacz czy Sw3 (prawy sw) jest naciśnięty
        jz      NoSw3
        cmp     Sw3.Pgdm, 0
        je      NoChanges
        mov     ax, 0
        mov     Left.Index, ax                  ;reset indeksów klawiszy dla wszystkich
        mov     Right.Index, ax                 ;klawiszy z wyjątkiem Sw1
        mov     Up.Index, ax
        mov     Down.Index, ax
        mov     Sw0.Index, ax
        mov     Sw2.Index, ax
        mov     Sw3.index, ax
        mov     bx, Sw3.Index
        mov     ax, Sw3.Index
        mov     bx, Sw3.Ptrs[bx]
        add     ax, 2
        cmp     ax, Sw3.Cnt
        jb      SetSw3
        mov     ax, 0
SetSw3: mov     Sw3.Index, ax
        call    PutStrInBuf
        jmp     NoChanges

NoSw3:  test    al, 10h                 ;zobacz czy Cooley został  naciśnięty w górę
        jz      NoUp
        cmp     Up.Pgmd, 0
        je      NoChanges
        mov     ax, 0
        mov     Right.Index, ax                 ;reset wszystkich przycisków w górze
        mov     Left.Index, ax
        mov     Down.Index, ax
        mov     Sw0.Index, ax
        mov     Sw2.Index, ax
        mov     Sw3.index, ax
        mov     bx, Up.Index
        mov     ax, Up.Index
        mov     bx, Up.Ptrs[bx]
```

```
                add     ax, 2
                cmp     ax, Up.Cnt
                jb      SetUp
                mov     ax, 0
SetUp:          mov     Up.Index, ax
                call    PutStrInBuf
                jmp     NoChanges

NoUp:           test    al, 20h                 ;zobacz czy Cooley został przesunięty w  lewo
                jz      NoLeft
                cmp     Left.Pgmd, 0
                je      NoChanges
                mov     ax, 0
                mov     Right.Index, ax         ;reset wszystkich klawiszy  w Lewo
                mov     Up.Index, ax
                mov     Down.Index, ax
                mov     Sw0.Index, ax
                mov     Sw2.Index, ax
                mov     Sw3.index, ax
                mov     bx, Left.Index
                mov     ax, Left.Index
                mov     bx, Left.Ptrs[bx]
                add     ax, 2
                cmp     ax, Left.Cnt
                jb      SetLeft
                mov     ax, 0
SetLeft:        mov     Left.Index, ax
                call    PutStrInBuf
                jmp     NoChanges

NoLeft:         test    al, 40h         ;zobacz czy Cooley został przesunięty w  lewo
                jz      NoRight
                cmp     Right.Pgmd, 0
                je      NoChanges
                mov     ax, 0
                mov     Left.Index, ax          ;reset wszystkich klawiszy  w Lewo
                mov     Up.Index, ax
                mov     Down.Index, ax
                mov     Sw0.Index, ax
                mov     Sw2.Index, ax
                mov     Sw3.index, ax
                mov     bx, Rightt.Index
                mov     ax, Right.Index
                mov     bx, Right.Ptrs[bx]
                add     ax, 2
                cmp     ax, Right.Cnt
                jb      SetRight
                mov     ax, 0
SetRight:       mov     Right.Index, ax
                call    PutStrInBuf
                jmp     NoChanges

NoRight         test    al., 80h                ;zobacz czy Cooley został przesunięty w dół
                jz      NoChanges
                cmp     Down.Pgmd, 0
                je      NoChanges
                mov     ax, 0
                mov     Left.Index, ax          ;reset wszystkich klawiszy  w Dó³
                mov     Up.Index, ax
```

```
                mov     Right.Index, ax
                mov     Sw0.Index, ax
                mov     Sw2.Index, ax
                mov     Sw3.index, ax
                mov     bx, Down.Index
                mov     ax, Down.Index
                mov     bx, Down.Ptrs[bx]
                add     ax, 2
                cmp     ax, Down.Cnt
                jb      SetDown
                mov     ax, 0
SetDown:        mov     Down.Index, ax
                call    PutStrInBuf


NoChanges:      pop     dx
                pop     bx
                pop     ax
                pop     ds
                jmp     cs: Int1Cvect
MyInt1c         endp
                assume  ds:nothing

;PutStrInBuf-   BX zawiera ciąg słów zakończonych zerem. Przesyłamy każde słowo przez
;               wywo³anie PutInBuffer

PutStrInBuf     proc    near
                push    ax
                push    bx
PutLoop:        mov     ax, [bx]
                test    ax. Ax
                jz      PutDone
                call    PutInBuffer
                add     bx, 2
                jmp     PutLoop

PutDone:        pop     bx
                pop     ax
                ret
PutStrInBuf     endp

;PutInBuffer-   znaki wyjściowe i kody znaków w AX do bufora roboczego

                assume  ds.:nothing
KbdHead         equ     word  ptr ds:[1ah]
KbdTail         equ     word  ptr ds:[1ch]
KbdBuffer       equ     word ptr ds:[1eh]
EndKbd          equ     3eh
Buffer          equ     1eh
PutInBuffer     proc    near
                push    ds
                push    bx
                mov     bx, 40h
                mov     ds, bx
                pushf
                cli                             ;to jest region krytyczny!
                mov     bx, KbdTail             ;pobranie wskaźnika końca bufora roboczego
                inc     bx                      ;i zrobienie miejsca na ten znak
                inc     bx
```

```
                cmp     bx, buffer+32           ;fizyczny koniec bufora?
                jb      NoWrap
                mov     bx, buffer              ;zawinięcie z powrotem do 1eh jeśli koniec

NoWrap:         cmp     bx, KbdHead             ;przepełnienie bufora?
                je      PIBDone
                xchg    KbdTail, bx             ;ustawiamy nowy, pobieramy stary wskaźnik
                mov     ds.:[bx], ax            ;dane wyjściowe AX do starej lokacji
PIBDone:        popf                            ;przywrócenie przerwań
                pop     bx
                pop     ds.
                ret
PutInBuffer     endp

;*****************************************************************************************
*;
; FindCode: Na wejściu, ES:DI wskazuje jakiś kod w * tym *programie który pojawia się w grze ATP. DS.:SI
;               wskazuje blok pamięci w grze XWING. FindCode przeszukuje całą pamięć aby znaleźć
;               podejrzany kawałek kodu i zwraca w DS.:SI wskazujący początek tego kodu. Kod ten zakłada
;               ,że * znajdzie * ten kod!. Zwraca wyzerowaną flagę przeniesienia jeśli znajdzie, ustawioną
;               jeśli nie.

FindCode        proc    near
                push    ax
                push    bx
                push    dx

DoCmp:          mov     dx, 1000h
CmpLoop:        push    di                      ;zachowujemy wskaźnik do kodu porównywanego
                push    si                      ;zachowanie wskaźnika do początku ciągu
                push    cx                      ;zachowanie licznika
        repe    cmpsb
                pop     cx
                pop     si
                pop     di
                je      FoundCode
                inc     si
                dec     dx
                jne     CmpLoop
                sub     si, 1000h
                mov     ax, ds
                inc     ah
                mov     ds, ax
                cmp     ax, 9000h
                jb      DoCmp

                pop     dx
                pop     bx
                pop     ax
                stc
                ret
FoundCode:      pop     dx
                pop     bx
                pop     ax
                clc
                ret
FindCode        endp
```

```
;*****************************************************************************
*;
; Podprogramy joysticka i przycisku, które pojawiają się w grze Xwing. Kod ten jest rzeczywistą daną ,
; ponieważ kod aktualizujący INT 21h przeszukuje całą pamięć dla tego kodu po załadowaniu pliku z dysku.

JoystickCode    proc    near
                sti
                neg     bx
                neg     di
                pop     bp
                pop     dx
                pop     cx
                ret
                mov     bp, bx
                in      al., dx
                mov     bl, al.
                not     al
                and     al, ah
                jnz     $+11h
                in      al, dx
JoystickCode    endp
EndJSC:

JoyLength       =       EndJSC-JoystickCode

ReadSwCode      proc
                mov     dx, 201h
                in      al, dx
                xor     al, 0ffh
                and     ax, 0f0h
ReadSwCode      endp
EndRSC:

ButtonLength    =       EndRSC-ReadSwCode

cseg            =       ends

Segemnt Instalacji

;Przesunęliśmy to tu aby nie zajmowała zbyt dużo miejsca w przestrzeni rezydentnej części tej łatki

DfltFCB         byte    3, „ „„ 0,0,0,0,0
CmdLine         byte    2, “ “, 0dh, 126  dup ( “ “)            ;linia poleceń dla programu
Pgm             byte    „XWING.EXE”, 0
                Byte    128  dup (?)                            ;dla nazwy użytkownika

; ChkBIOS15-    Sprawdzamy aby zobaczyć czy sterownik INT 15 dla FSPro jest obecny w pamięci

ChkBIOS15       proc    far
                mov     ah, 84h
                mov     dx, 8100h
                int     15h
                mov     di, bx
                strcmpl
                byte    “CH Products: Fightstick Pro”, 0
                jne     NoDriverLoaded
                ret

NoDriverLoaded:
```

```
                print
                byte    “Sterownik CH Products SGDI dla FlightStick Pro nie jest “
                byte    “załadowany do pamięci.”, cr, lf
                byte    „Proszę uruchomić FSPSGDI przed uruchomieniem tego programu”
                byte    cr, lf, 0
                exitpgm
ChkBIOS15       endp

;*****************************************************************************************
*
;
; Identify-     Drukowanie zapisanej wiadomości

                assume  ds.:nothing
Identify        proc    far

;Drukujemy ciąg powitalny. Zauważmy ,że ciąg „VersionStr” będzie zmodyfikowany przez  program
; „version.exe” za każdym razem, kiedy zasemblujemy ten kod

                print
                byte    cr, lf,lf
                byte    “X W I N G P A T C H” ,cr ,lf
                byte    „CH Products Flightstick Pro”, cr, lf
                byte    Copyright 1994, Randall Hyde”, cr, lf
                byte    lf
                byte    0

                ret
Identify        endp

;*****************************************************************************************
*
;
; Kalibrowanie d³awienia

                assume  ds:nothing
Calibrate       proc    far
                print
                byte    cr, lf, lf
                byte    “Kalibracja:”, cr, lf, lf
                byte    „Move the throttle to one extreme and press any „
                byte    “button:’, 0

                call    Wait4Button
                mov     ah, 84h
                mov     dx, 1h
                int     15h
                push    dx                                      ;zachowanie odczytu potencjometru 3

                print
                byte    cr, lf
                byte    „Move the throttle to the other extreme and press „
                byte    “any button:”, 0
                call    Wait4Button
                mov     ah, 84h
                mov     dx, 1
                int     15h
                pop     bx
                mov     ax, dx
```

```
                cmp     ax, bx
                jb      RangeOkay
                xchg    ax, bx
RangeOkay:      mov     cx, bx                  ;obliczanie wartości centralnej
                sub     cx, ax
                shr     cx, 1
                add     cx, ax
                mov     ah, 84h
                mov     dx, 303h                ;kalibracja potencjometru trzy
                int     15h
                ret
Calibrate       endp

Wait4Button     proc    near
                mov     ah, 84h                 ;najpierw czekamy aż wszystkie przyciski
                mov     dx, 0                   ;zostaną zwolnione
                int     15h
                and     al., 0F0h
                cmp     al., 0F0h
                jne     Wait4button

                mov     cx, 0
Delay:          loop    Delay

Wait4Press:     mov     ah, 1                   ;zjadany znak z klawiatury który przychodzi
                int     16h                     ;i obsługa właściwa ctrl-C
                je      NoKbd
                getc

NoKbd:          mov     ah, 84h                 ;teraz oczekujemy na naciśnięcie jakiegoś
                mov     dx, 0                   ;przycisku
                int     15h
                and     al., 0F0h
                cmp     al.,0F0h
                je      Wait4Press
                ret
Wait4Button     endp
Instalation     ends

sseg            segment para stack 'STACK'
                word    256  dup (0)
endstk          word    ?
sseg            ends

zzzzzzseg       segment para public 'zzzzzzseg'
Heap            byte    1024   dup  (0)
zzzzzzseg       ends
                end     Main
```

24.8 PODSUMOANIE

Karta złącza gier PC pozwala nam na podłączenie szerokiej gamy urządzeń wejściowych do naszego PC. Do urządzeń takich zaliczamy cyfrowe joysticki, **paddles,** joysticki analogowe, **steering wheels**,**yokes** i inne. **Paddle** dostarcza jednego stopnia swobody, joysticki dostarczają dwóch stopni swobody wzdłuż osi (X,Y). **Steering wheels** i **yokes** również dostarczają dwóch stopni swobody, chociaż są zaprojektowane dla innych typów gier.

*"Typowe urządzenia gier"

Większość wejściowych urządzeń gier jest podłączonych do PC przez kartę złącza gier. Urządzenie to dostarcza do czterech cyfrowych (przełączników) wejść i czterech analogowych (rezystywnych) wejść. Urządzenie to pojawia się jako pojedyncza lokacja I/O w przestrzeni adresowej PC. Cztery z tych bitów pod tym portem odpowiada czterem przełącznikom, cztery z tych wejść dostarcza stanu impulsu zegarowego z chipu 558 dla wejścia analogowego. Przełączniki możemy odczytać bezpośrednio z portu; aby odczytać wejście analogowe musimy stworzyć pętlę czasową zliczającą jak długo pobierany jest impuls powiązany z poszczególnym urządzeniem idąc od wartości największej do najmniejszej

*"Sprzętowe złącze gier"

Oprogramowanie złącza gier byłoby prostym zadaniem z wyjątkiem tego, że będziemy pobierać różne odczyty dla tych samych pokrewnych pozycji potencjometrów z różnymi kartami złącza gier, urządzeniami wejściowymi gier, systemami komputerowymi i oprogramowaniem. Prawdziwą sztuczką przy programowaniu złącza gier jest stworzenie spójnych wyników bez względu na aktualnie używany sprzęt. Jeśli możemy żyć z niezmodyfikowanymi wartościami wejściowymi, BIOS dostarcza dwóch funkcji do odczytu przełączników i wejść analogowych. Jednakże jeśli musimy znormalizować wartości, prawdopodobnie będziemy musieli napisać swój własny kod. Jednak napisanie takiego kodu jest bardzo łatwe jeśli pamiętamy o podstawowych zasadach algebry.

*"Użycie funkcji I/O gier BIOS"
*"Pisanie własnego podprogramu gier I/O"

Podobnie jak z innymi urządzeniami w PC, jest problem z bezpośrednim dostępem do sprzętowego złącza gier, taki kod nie działa ze sprzętem, który nie stosuje ściśle oryginalnych kryteriów projektowania PC. Wymyślne urządzenia wejściowe gier takie jak joystick Thrustmaster i CH Product's FlightStick Pro będą wymagały napisania specjalnego sterownika programowego . Co więcej, nasz podstawowy kod joysticka może nawet nie działać z przyszłymi urządzeniami, nawet jeśli dostarczą one minimalnego zbioru cech kompatybilnych ze standardowymi urządzeniami gier. Niestety, usługi BIOS są bardzo wolne i niezbyt dobre., więc niewielu programistów wykonuje wywołania BIOS, pozwalając projektantom dostarcza wymiennych sterowników dla sowich urządzeń gier. Dla złagodzenia tego problemu rozdział ten przedstawia aplikację Standard Game Device Input z interfejsem programowalnym – zbiór funkcji specjalnie zaprojektowanych dla dostarczenia rozszerzalnego, przenośnego systemu dla urządzeń wejściowych. Bieżąca specyfikacja dostarcza do 256 cyfrowych i 256 analogowych urządzeń wejściowych i jest łatwe rozszerzenie obsługi urządzeń wyjściowych i innych urządzeń wejściowych.

*"Standardowy Interfejs Urządzenia Gier(SGDI)
*"Interfejs Programowy Aplikacji"

Rozdział ten kończy się przykładem półrezydentnego programu, który wykonuje wywołanie SGDI. Pogram ten, który aktualizuje popularną grę Xwing, dostarcza pełnego wsparcia dla CH Product's FlightStick Pro w Xwing. Program ten demonstruje wiele cech sterownika SGDI również dostarczając przykładu jak załatać dostępną komercyjną grę

*"Aktualizacja istniejących gier"