Kuchnia INFORMIKA

Roland Wacławek

Można sądzić, że mikrokomputer stał się już stałym elementem naszej codzienności. Dotyczy to zwłaszcza sprzętu klasy PC. Zagościł w biurach i zakładach, coraz częściej bywa też w domu. Dostępne jest oprogramowanie i związana z nim fachowa literatura. Szczególnie uprzywilejowane są zastosowania biurowe, poczynając od przetwarzania tekstów i technik wydawniczych, a kończąc na rozmaitych kalkulacjach. Utrwalił się stereotyp PC na biurku, wspomagającego prace typowo umysłowe. Nie doceniane są natomiast inne obszary zastosowań PC, a jest to przecież narzędzie niesłychanie uniwersalne. Dziedziną szczególnie zaniedbaną jest wymiana danych z otoczeniem.

Przy odrobinie inwencji i know-how PC może stać się wspaniałym pomocnikiem w każdej pracowni elektronicznej, fotograficznej, modelarskiej. Może także usprawnić i uprzyjemnić nam rozmaite dziedziny życia domowego, poczynając od błahostek, jak opieka nad akwarium czy kolejką elektryczną, a kończąc na sprawach życiowo istotnych, jak inteligentny system przeciwwłamaniowy lub racjonalizacja zużycia energii. Wszystkich zastosowań wymienić nie sposób. Okazuje się przy tym najczęściej, że niezbędne nakłady materialne są bardzo niewielkie.

Wiele starszych komputerów, technicznie sprawnych, wychodzi dziś z „pierwszej linii" z powodu zużycia moralnego. Czy mają trafić na złom lub stopniowo pokrywać się kurzem? Każdy, nawet najprostszy, stary PC może zostać doskonałym sterownikiem, stróżem, przyrządem pomiarowym, analizatorem sygnałów, domowym centrum automatyki. Rozpoczynamy cykl artykułów adresowanych do użytkowników PC(AT)... o twórczym, prawdziwie inżynierskim zacięciu. Takich, którzy lubią eksperymenty i nie dostają wysypki patrząc na wkrętak lub lutownicę. Chcemy zaproponować odkrywanie PC w inny sposób niż poprzez WINDOWS czy dBASE – postaramy się stopniowo odsłaniać jego „prawdziwe oblicze". Będziemy krążyć wokół nietypowych sposobów wykorzystania PC, tak pod względem sprzętowym jak i programowym. Spróbujemy dostarczyć wiadomości technicznych niezbędnych do własnych poszukiwań, lecz trudno dostępnych w innych źródłach. Pokażemy sztuczki i chwyty tak sprzętowe, jak i programowe, chociażby celem było tylko lepsze poznanie komputera. Dalszą tematykę cyklu będziecie – mamy taką nadzieję – współtworzyć wspólnie z nami, nasi Drodzy Czytelnicy. Dzisiaj natomiast omówimy ogólne problemy „przerzucania" danych między procesorem a jego interfejsami.

Łączyć się z PC można albo wykorzystując istniejące w nim już standardowe interfejsy, albo też konstruując własne, specjalizowane karty, wkładane wprost do gniazd magistrali systemowej. Żadna z tych technik nie przekracza możliwości nawet średnio zaawansowanego amatora-elektronika. W wielu przypadkach wystarczy umiejętnie skorzystać z już wmontowanych w komputer możliwości. Nasze poszukiwania zaczniemy rzecz jasna od technik nie wymagających otwierania obudowy komputera. Ortodoksja w myśleniu prowadzi do niedorzecznego przekonania, że np. interfejs równoległy może służyć tylko do podłączenia drukarki. Czy tak jest naprawdę, przekonamy się wkrótce.

Ogromna większość komputerów klasy PC ma „od urodzenia" przynajmniej po jednym interfejsie równoległym i szeregowym oraz dla joysticków (*game port*). Interfejsy te są zazwyczaj zgrupowane na jednej karcie, tzw. wielofunkcyjnej (*multifunction* lub *multi I/O*). Prawie wszystkie karty tego typu zawierają 2 interfejsy szeregowe (RS232C). Jeżeli jeden jest nieaktywny, można temu zaradzić, obsadzając przygotowane na karcie podstawki odpowiednimi układami scalonymi. Komputery z kartą graficzną HGC (Hercules) mają często po dwa interfejsy równoległe: jeden na karcie HGC, drugi na karcie wielofunkcyjnej. Miejsc, do których można się podłączyć bez zaglądania do wnętrza maszyny jest więc sporo.

Komunikacja z interfejsami odbywa się za pośrednictwem rejestrów wejścia-wyjścia, inaczej: portów. W mowie ojczystej termin: port jest językowo niefortunny, ale poręczny, gdyż powszechnie się przyjął; pozostańmy przy nim. Porty dzielimy na wyjściowe (wy, ang. *output*, w skrócie: O) i wejściowe (we, ang. *input*, w skrócie: I). Porty wyjściowe służą do wyprowadzania informacji na zewnątrz komputera, np. w postaci odpowiedniego sygnału elektrycznego na złączu interfejsu. Porty wejściowe dostarczają informacji o stanie otoczenia komputera, np. o stanie elektrycznym wybranych końcówek złącza.

W zabawnym uproszczeniu rejestr wyjść jest komórką pamięci z wyprowadzonymi do poszczególnych bitów drucikami, tak że jej zawartość można odczytać woltomierzem. W praktyce nie zawsze „druciki" muszą był dosłownie wyprowadzone na zewnątrz komputera – mogą np. służyć do wewnętrznego sterowania trybami pracy interfejsu. Termin: „na zewnątrz" odnosi się tu do wszystkiego, co nie jest bezpośrednio dołączone do magistrali centralnego procesora. Nie wszystkie bity rejestru muszą być wykorzystane. Wartości wpisywane do bitów nie wykorzystanych nie mają wówczas żadnego znaczenia.

Spośród rejestrów wejścia także nie każdy dostarcza danych wprost do gniazda. Wiele rejestrów podaje dane np. z kart graficznych, sterowników dysków lub zegara systemowego. Także i tu nie wszystkie bity muszą być wykorzystane. Wartości pozostałych bitów są wówczas stałe albo zupełnie przypadkowe; zależy to od technicznych szczegółów konstrukcji interfejsu.

W komputerach PC porty mogą być 8- lub 16-bitowe. Oznacza to, że w pojedynczej operacji komputer może zapisać bądź odczytać „porcję" 8 (bajt) lub 16 (słowo) bitów. W praktyce spotykamy rejestry 8-bitowe. Dlaczego? Protoplasta rodu, PC z procesorem 8088, miał magistralę 8-bitową i dla takiej magistrali zaprojektowano interfejsy, które z czasem nabrały mocy standardu. Organizację 8-bitową ma poza tym ogromna większość produkowanych scalonych układów we/wy. Nowoczesne komputery mają magistralę 16-bitową (AT), a nawet 32-bitową (np. EISA). Aby umożliwić stosowanie tych samych kart dodatkowych (ang. *add-on card*) przewidziano jednak możliwość pracy tych magistral w trybie 8-bitowym. Pozwala to np. użyć tej samej karty graficznej zarówno w PC, jak i w 486. Oczywiście 8-bitowy tryb operacji wejścia/wyjścia zwalnia pracę szybkiego systemu, ale w praktyce operacji tych jest stosunkowo niewiele, tak że efekt końcowy jest niezauważalny.

Jak odwołujemy się do portów? Podobnie jak komórki pamięci, porty mają swoje adresy. Procesory 8088... itd. dopuszczają 16-bitowe adresy portów od 0 do 65535 ($=2^{16}-1$). Z powodów konstrukcyjnych w komputerach klasy PC do adresowania portów wykorzystuje się tylko 10 bitów, a zatem dopuszczalne adresy portów ograniczają się do przedziału 0 ... 1023 ($=2^{10}-1$). Daje to 1024 portów wejściowych i tyleż wyjściowych. W praktyce ograniczenie tej tzw.

przestrzeni adresowej I/O nie jest bolesne, natomiast upraszcza konstrukcję kart dodatkowych. Sami skorzystamy z tego dobrodziejstwa. W języku maszynowym procesora są specjalne rozkazy do komunikacji z portami: **IN** (wczytaj) i **OUT** (zapisz). Posługiwanie się językiem maszynowym to jednak ostateczność. Wiele języków wysokiego poziomu udostępnia mechanizmy wystarczająco bliskie sprzętu. Należy do nich TURBO–Pascal, niekwestionowany lider wśród popularnych systemów programowania PC. Ten wygodny w użyciu język posłuży nam za podstawowe narzędzie programowe. Przenoszenie programów pascalowych np. do języka C nie przedstawia problemów. Zakładamy, że podstawy TURBO–Pascala są zainteresowanym Czytelnikom znane, bądź mogą być opanowane bez większych problemów (kurs Pascala prowadził w swoim czasie także „Młody Technik"). Tutaj przypomnimy tylko mechanizmy nie spotykane w zwykłym programowaniu, a dla nas mające podstawowe znaczenie.

Programując na pograniczu sprzętu nie unikniemy kontaktu z systemem dwójkowym bądź szesnastkowym. W odróżnieniu od dziesiętnego pozwalają one bowiem „widzieć" w zapisie każdy bit z osobna. System szesnastkowy ma w TURBO Pascalu specjalną notację. Stała zaczynająca się znakiem '**$**' oznacza liczbę szesnastkową. Tak np. zapis: **$9F** oznacza dwójkowo 10011111, a dziesiętnie: 159. Bit o wartości 1 nazywamy ustawionym, bit 0 – skasowanym. Wskazując poszczególne bity powołujemy się zwykle na ich numery. Powszechnie przyjęta w świecie PC numeracja zakłada, że bit najmniej znaczący (najmłodszy) nosi numer 0. Przechodząc „w lewo", tzn. do bitów bardziej znaczących, napotykamy kolejne wyższe numery. Przykład: w bajcie 01001000 ustawione są bity nr 3 i 7.

W TURBO Pascalu dostęp do portów odbywa się jak do tablicy. Istnieją dwie wstępnie zadeklarowane tablice **Port** i **PortW**. Tablica: **Port** składa się z elementów 8-bitowych, **PortW**: 16-bitowych. Oczywiście tablice te nie mają żadnej reprezentacji w pamięci, w szczególności nie buforują one zapisywanych do portów wartości (wyjaśnimy to za chwilę). Obie tablice stanowią tylko konstrukcję językową. Jeżeli w programie pascalowym wystąpi np. instrukcja:

```
X:= Port[$300]; Port[$301]:= Y;
```

w programie maszynowym będą jej odpowiadać odpowiednie rozkazy **IN** i **OUT**. Różnica między tablicami **Port** i **PortW** polega tylko na tym, że przy odwołaniu do **Port** kompilator wygeneruje 8-bitowy, do **PortW**: 16-bitowy wariant rozkazu **IN** lub **OUT**. Instrukcja:

```
X:= PortW[$300]
```

spowoduje odczytanie 16-bitowej liczby z portu 300H. Stop! Wspomnieliśmy przed chwilą, że prawie wszystkie rejestry są 8-bitowe? Co zatem odczyta powyższy rozkaz w AT? A w XT, z jego 8-bitową magistralą?

Sterownik magistrali systemowej w PC po odwołaniu do konkretnego adresu otrzymuje niezwłocznie informację, w jakim formacie będą przesyłane dane. Jak – omówimy później, przy okazji zawierania bliższej znajomości z magistralą PC i AT. Jeżeli format danych jest ośmiobitowy, sterownik magistrali automatycznie dzieli operację na 2 fazy i np. zamiast jednego odczytu 16-bitowego spod adresu 300H wykona kolejno 2 odczyty 8-bitowe: jeden spod adresu 300H, drugi – spod adresu 301H. Następnie oba odczytane bajty zostaną poskładane w jedno słowo. Pierwszy bajt stanie się jego mniej znaczącą (młodszą), drugi bajt – bardziej znaczącą (starszą) częścią. Podobnie funkcjonują odwołania do pamięci.

Powyższy, pozornie oczywisty, mechanizm zmiany formatu odwołań do portów funkcjonuje jednak pewnie tylko w jedną stronę. Przypuśćmy oto, że port 16-bitowy pod adresem np. 300H odczytujemy w dwóch cyklach 8-bitowych. Pierwsze odwołanie – pod adres 300H – spowoduje wysłanie przez urządzenie do procesora całych 16 bitów, z których ten ostatni przyjmie tylko 8 najmłodszych. Drugie odwołanie, pod adres 301H, może nie spowodować wysłania przez procesor żadnych danych, gdy adres 301H nie zostanie rozpoznany przez urządzenie! Także i tu wszystko zależy od szczegółów konstrukcyjnych danego interfejsu. Na szczęście dzięki generalnie 8-bitowemu formatowi portów powyższy problem wystąpi rzadko. Myśląc o pamięci operacyjnej domniemamy pod danym adresem jedną jedyną komórkę, tę samą przy zapisie i odczycie. Z portami sprawa jest odmienna. Jeżeli do komórki RAM wpiszemy liczbę, możemy spodziewać się, że przy odczycie dostaniemy tę samą wartość. Załóżmy, że do portu wyjścia o danym adresie wpiszemy daną, a następnie spróbujemy odczytać ten port. Co uzyskamy? To już zależy od konstrukcji karty. Może same jedynki, może zera albo wartości przypadkowe; zależy to od szczegółów konstrukcyjnych portu. Port to nie komórka pamięci! Jeżeli pod tym samym adresem znajduje się port wejścia, to oczywiście przy odczycie otrzymamy jakieś sensowne dane. Nie muszą to jednak być wcale te dane, które wpisaliśmy. Wszystko zależy od tego, do czego są dołączone „druciki" portu wejściowego.

Niektóre porty wyjściowe są skojarzone z umieszczonym pod tym samym adresem portem wejściowym. Pozwala to odczytać wartość ostatnio wpisaną do portu (z dokładnością do nie używanych bitów). Nie jest to jednak reguła. Przeoczenie tego faktu może być przykre, zwłaszcza że zawartość portów traktujemy najczęściej jako kompozycję niezależnych bitów–przełączników, z których każdy steruje inną funkcją urządzenia. Normalny port można bowiem zapisywać tylko w całości, nadając nową wartość od razu wszystkim bitom. Jeżeli część bitów ma zachować swój stan, trzeba nadać im ponownie ich bieżące wartości. Trzeba zatem znać te wartości. Chcąc ustawić wybrany bit w komórce pamięci bez zmiany innych bitów, tworzymy sumę logiczną bieżącej zawartości komórki z inną liczbą dwójkową z jednym ustawionym bitem. Podobnie, chcąc bit skasować, tworzymy iloczyn logiczny zawartości z liczbą z jednym skasowanym bitem (reszta ustawiona). Przykład: ustawiamy bit nr 2 i kasujemy bit nr 7 w zmiennej **Alfa**. Zamiast „ręcznie" negować poszczególne bity, można użyć operatora logicznego NOT, który zaneguje wszystkie bity argumentu.

```
VAR Alfa: BYTE;
.....
Alfa:= Alfa AND NOT($80) or $04;
```

Mając do czynienia z portami wyjściowymi bez zwrotnego odczytu trzeba równolegle z zapisem do portu przechować zapisywaną wartość w pamięci, w tzw. zmiennej buforowej. Później można jej użyć jako odniesienia. Przykład: najpierw wpisujemy do portu 300H jakąś wartość, później chcemy dodatkowo ustawić w nim bity 0 i 1:

```
VAR B300: BYTE
.....
B300:= WartX; Port[$300]:=B300;
.....
B300:=B300 OR $3; Port[$300]:=B300;
```

Zmienna buforowa: B300 reprezentuje w tym systemie zawsze ostatnią wartość wpisaną do portu 300H.

Oprócz portów będzie nam potrzebny dostęp do pamięci operacyjnej, i to do komórek o ściśle określonych adresach. W procesorach 8088/86 przy odwołaniach do pamięci operacyjnej stosowany jest 20-bitowy adres, komponowany z szesnastobitowego **przesunięcia** (ang. *offset*) oraz również szesnastobitowego **segmentu**, pobieranego z jednego z rejestrów segmentowych. Kompozycja ta powstaje następująco:

```
adres efektywny:= (segment*16)+offset
```

Dwuczłonowość adresu wymaga specjalnej notacji. Podając pełny adres oddzielamy segment od przesunięcia dwukropkiem. Tak np. zapis **40H:011EH** oznacza komórkę o absolut-

nym, efektywnym adresie: 51EH. Warto zauważyć, że ten sam adres efektywny można zapisać na wiele sposobów, w naszym przypadku mogłoby to być z powodzeniem 50H:01EH lub 0:51EH. TURBO Pascal pozwala na absolutną adresację pamięci, udostępniając w tym celu specjalną konstrukcję w postaci wstępnie zdefiniowanych „tablic" **Mem** i **MemW**. Pierwsza pozwala operować na bajtach, druga na słowach 16-bitowych. Ich osobliwością jest dwuczłonowy indeks, ściśle odpowiadający systemowi adresacji pamięci. Przykład:

```
Tryb_graficzny:= Mem[$40:$49];
Znakow_w_linii:= MemW($40:$4A];
```

Powyższe instrukcje są całkiem sensowne, odczytują bowiem wprost z obszaru parametrów BIOS numer bieżącego trybu graficznego (1 bajt) i liczbę znaków w linii (liczba 2-bajtowa). Elegantszy i wygodniejszy sposób odwołań do absolutnych adresów pamięci polega na użyciu w deklaracji zmiennej klauzuli: **ABSOLUTE**. Powoduje ona, że kompilator zamiast przydzielać dla zmiennej obszar pamięci potraktuje podany adres tak, jak gdyby znajdowała się tam zmienna określonego typu. Jeśli ktoś lubi smerfowe efekty, niech skompiluje poniższy program do pliku .EXE i uruchomi z poziomu DOS:

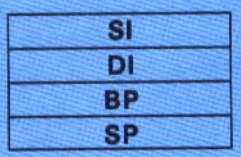
```
VAR Znakow_w_linii:WORD ABSOLUTE $40:$4A;
BEGIN Znakow_w_linii:= 81
END.
```

Programy aplikacyjne tylko w wyjątkowych przypadkach odwołują się wprost do portów. Normalnie pomostem pośredniczącym w wymianie danych z interfejsami są procedury zawarte w pamięci stałej BIOS, a wywoływane za pomocą tzw. przerwań programowych, będących w gruncie rzeczy wywołaniami podprogramów o adresach zapisanych w z góry ustalonych miejscach w pierwszych 1024 komórkach pamięci. Pewnym wyjątkiem są interfejsy szeregowe, obsługiwane przez BIOS bardzo prymitywnie. Dlatego programy korzystające z transmisji szeregowej, jak emulatory terminali czy chociażby zwykły drajwer (ang. *driver*) myszy, na ogół biorą całą obsługę interfejsu szeregowego na siebie i oczywiście komunikują się z nim na poziomie portów.

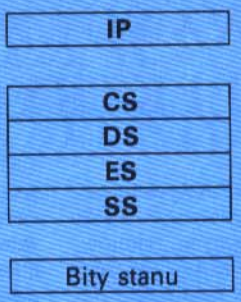
Zamierzamy unikać w miarę możliwości języka maszynowego. Mówiąc o przerwaniach nie sposób jednak całkiem pominąć architektury procesora. Przerwanie programowe polega w końcu na wywołaniu pewnego podprogramu maszynowego, przy czym przekazywanie danych odbywa się, jak zwykle w takich przypadkach, w rejestrach procesora. Samo ładowanie rejestrów procesora i wywołanie przerwania wraz z towarzyszącym temu rytuałem przechowywania, a następnie odtwarzania zawartości rejestrów na stosie bierze na siebie TURBO Pascal. Do nas należy jednak określenie zawartości rejestrów w chwili wywołania, a następnie ich interpretacja po powrocie z przerwania. Z jakimi więc rejestrami mamy do czynienia?

Niezależnie od procesora rzeczywiście wbudowanego w komputer TURBO Pascal zakłada model rejestrów 8088/86. Jest to sensowne, gdyż procedury BIOS działają w trybie adresów rzeczywistych, w którym procesory od 286 do 486 są zgodne programowo z 88/86:

Rejestry ogólnego przeznaczenia



Rejestry adresowe



Wskaźnik rozkazu

Rejestry segmentowe

Rejestr flagowy

Rejestry ogólnego przeznaczenia służą głównie do przechowywania danych i do wykonywania na nich operacji. Unikalny dla tej grupy jest fakt, że można je traktować zarówno jako rejestry ośmiobitowe (np. **AH, AL**), jak i łączyć w pary, gdzie funkcjonują w formacie szesnastobitowym (np. **AX**). Przy łączeniu **AH, BH, CH** i **DH** tworzą zawsze starszą połówkę rejestru szesnastobitowego, pozostałe rejestry tej grupy – młodszą połówkę.

Szesnastobitowe rejestry adresowe są wykorzystywane głównie do przechowywania przesunięć adresowych komórek pamięci operacyjnej. SP jest wskaźnikiem stosu procesora i stanowczo nie należy go modyfikować.

Funkcja rejestrów segmentowych wiąże się ściśle z omówionym modelem pamięci operacyjnej procesora 88/86. Wszelkie adresy pamięci są wyznaczane zawsze z udziałem któregoś z tych rejestrów. Przesunięcie adresowe jest zawarte wprost w rozkazie albo brane z któregoś z rejestrów adresowych lub z **IP**. Rejestr flagowy interpretujemy jako kompozycję niezależnych bitów, z których każdy sygnalizuje określony element stanu procesora, np. zerowy wynik ostatniej operacji arytmetycznej. Kluczem do przerwań programowych w TURBO Pascalu jest typ rekordowy REGISTERS i procedura **Intr**. Obiekty te są wstępnie zdefiniowane w module DOS.TPU:

```
TYPE REGISTERS=
  RECORD
    CASE INTEGER OF
      0: (AX,BX,CX,DX,
          BP,SI,DI,DS,ES,Flags: Word);
      1: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
    END;

PROCEDURE Intr(IntNo: BYTE; VAR Regs: REGISTERS);
```

Rekord reprezentuje blok komórek pamięci, które w chwili wywołania procedury **Intr** zostaną skopiowane do odpowiednich rejestrów procesora, po czym procedura **Intr** wywoła przerwanie programowe o numerze zakresu 0...255, podanym jako pierwszy parametr. Po powrocie z przerwania procedura **Intr** natychmiast przepisze zawartości rejestrów do odpowiednich pól rekordu, czyniąc je dostępnymi z poziomu Pascala. Rejestry: **SP, IP, CS** i **SS** nie mają reprezentacji w rekordzie i ma to sens. Dostęp do tych rejestrów jest zbędny, gdyż nigdy nie uczestniczą one w wymianie parametrów. Natomiast praktycznie każda zmiana ich zawartości „zawiesza" system. Dla ilustracji wyprowadzimy znak na drukarkę korzystając wprost w usług BIOS. Za drukarkę odpowiada przerwanie 17H, aby wydrukować znak należy do rejestru **AH** wpisać 0, do **DX** – numer drukarki, natomiast do **AL** – kod znaku. Jeżeli po powrocie bity nr 3 i 5 w AH nie są skasowane, a bit nr 4 – ustawiony, oznacza to błąd:

```
USES DOS;
VAR Rejestry: REGISTERS;
BEGIN WITH Rejestry DO
  BEGIN AH:= 0; DX:= 0; {LPT1}
    AL:= 65;        {kod litery 'A'}
    Intr ($17, Rejestry);
    IF (AH AND $38) <> $10 THEN
      Writeln ('Usterka drukarki');
  END;
END.
```