

Opracowanie wersji elektronicznej:
Mariusz "Mario" Bielak

Janusz B. Wiśniewski

Zestaw do programowania w języku assemblera 6502 dla ATARI XL/XE

Podręcznik użytkownika

Copyright © 1991 Laboratorium Komputerowe A V A L O N

Niniejszy podręcznik stanowi komplet z nośnikiem magnetycznym (kasetą lub dyskietką) zawierającym zestaw programów do programowania komputerów ATARI serii XL i XE w języku assemblera. Oprócz omówienia tych programów zawiera także zwięzły opis rozkazów procesora 6502 oraz zasady programowania w języku assemblera, może więc służyć nie tylko zaawansowanym programistom, lecz także zupełnym nowicjuszom.

Uwaga!

Przed rozpoczęciem pracy z pakietem należy sporządzić kopię roboczą zakupionego zestawu. Wszystkie pliki z kasyety lub dyskietki należy przenieść na taśmę bądź dysk, na którym będziemy pracować. Zakupione oprogramowanie należy przechowywać w bezpiecznym miejscu jako kopię bezpieczeństwa.

Wszelkie uszkodzenia oryginalnego nośnika informacji powstałe w wyniku zaniedbania powyższej uwagi nie będą rozpatrywane na prawach reklamacji.

Okładka książki, wkładka do kasyety, koperta dyskietki:
kompozycja graficzna: Danuta Sienkowska,
zdjęcie: Janusz B. Wiśniewski.

Spis treści

| | |
|----------------------------------------------------------|-----------|
| WSTĘP | 5 |
| PROGRAMOWANIE PROCESORA 6502 | 7 |
| Pamięć, adresy, rejestry | 7 |
| Budowa rozkazu | 8 |
| Tryby adresacji | 8 |
| Strona zerowa | 9 |
| Stos | 9 |
| Znaczniki | 10 |
| Rozkazy procesora 6502 | 11 |
| Rozkazy przesłań, arytmetyczne i logiczne | 11 |
| Rozkazy operacji na stosie i/lub wskaźniku instrukcji .. | 13 |
| Rozkazy wewnętrzne procesora | 14 |
| JĘZYK ASEMBLERA | 16 |
| Sposób pracy assemblera, licznik lokacji | 16 |
| Liczby, etykiety, wyrażenia | 16 |
| Format rozkazu | 17 |
| Pseudorozkazy, definiowanie danych | 19 |
| OPT | 19 |
| ORG | 20 |
| EQU | 21 |
| END | 21 |
| ICL | 21 |
| DTA | 22 |
| Różne drobne i grube uwagi | 23 |
| Błędy asemblacji | 24 |
| QUICK ASSEMBLER | 25 |
| File | 26 |
| Load i Save | 26 |
| New | 28 |
| Dir | 29 |

| | |
|--------------------------------------|-----------|
| Prn i Obj | 29 |
| Quit | 30 |
| Edit | 30 |
| Informacje o stanie edytora | 31 |
| Redagowanie tekstu | 32 |
| Run | 35 |
| Assembly | 37 |
| Setup | 38 |
| Ws mode | 38 |
| Run | 39 |
| Memhi | 39 |
| Save Setup | 39 |
| Qasm | 40 |
| BUG HUNTER | 41 |
| File | 42 |
| Edit | 44 |
| Trace | 46 |
| Break | 47 |
| Setup | 49 |
| XL FRIEND | 51 |
| Edytor | 52 |
| Tabela kodów | 54 |
| Kalkulator | 54 |
| Monitor pamięci | 54 |
| TEKSTY ŹRÓDŁOWE PROGRAMÓW | 56 |
| Zestawy standardowych procedur | 56 |
| _IO | 57 |
| _PM | 59 |
| _SOUND | 60 |
| Programy przykładowe | 62 |
| TRANS | 62 |
| DEMO | 63 |

WSTĘP

Drogi Użytkowniku! Zostałeś posiadaczem zestawu do programowania w języku assemblera. W skład kompletu wchodzi dyskietka lub kaseeta, zawierająca programy narzędziowe, oraz niniejszy podręcznik. Ponieważ z oczywistych względów drukowanie podręcznika trwa długo, należy się spodziewać w oferowanym zestawie pewnych modyfikacji zawartości, będących naturalnym skutkiem nigdy nie zamierającego postępu. Wszystkie takie zmiany, dostrzeżone błędy i uzupełnienia podręcznika są opisane bezpośrednio na dyskietce (kasecie). Należy koniecznie się z nimi zapoznać, uruchamiając na wstępie program "CZYTAJ".

Programy są opisywane głównie pod kątem współpracy z napędem dysków elastycznych, ale także posiadacz magnetofonu nie będzie miał z nimi kłopotów. W tych miejscach opisów, gdzie sposób postępowania istotnie się różni, będzie to omówione. Fragmenty dotyczące tylko jednego z nośników są oznaczone pionową kreską wzdłuż lewej krawędzi tekstu i literą **D** (dyskietka) lub **C** (taśma).

D Dyskietkę należy umieścić w napędzie, etykietą do góry, a następnie włączyć komputer, trzymając wciśnięty klawisz OPTION. Zostanie wówczas wczytany DOS (dyskowy system operacyjny), który zamelduje się napisem "D1:". Przy pierwszym obcowaniu z dyskietką należy napisać:

CZYTAJ

i nacisnąć klawisz RETURN, co spowoduje uruchomienie programu "CZYTAJ.COM", wyświetlającego najświeższe (ważne) informacje, uzupełniające niniejszy podręcznik. W szczególności zawarto tam zasady korzystania z DOS-u.

C Kasetę, przewiniętą na początek strony A, należy umieścić w magnetofonie, następnie zaś włączyć komputer, trzymając wciśnięte klawisze START i OPTION. Po usłyszeniu sygnału dźwiękowego przeprowadzić operację wczytywania zgodnie z zasadami obsługi posiadanego magnetofonu. Uruchomiony wówczas zostanie COS (kasetowy system operacyjny), który zamelduje się znakiem ">". Przy pierwszym obcowaniu z kasetą należy napisać:

i nacisnąć klawisz RETURN (kaseta ustawiona w miejscu, gdzie zakończyło się czytanie COS-u), co spowoduje wczytanie i uruchomienie programu "CZYTAJ.COM", wyświetlającego najświeższe (ważne) informacje, uzupełniające niniejszy podręcznik. W szczególności zawarto tam zasady korzystania z COS-u.

Uwaga: w opisach klawisz CONTROL oznaczany bywa jako " ^ ", zaś klawisz SHIFT jako " ! ".

Wszelkie uwagi i propozycje można nadsyłać na adres

Laboratorium Komputerowe
AVALON
38-100 Strzyżów
skr.poczt. 46

W przypadkach wymagających odpowiedzi prosimy załączyć opłaconą kopertę zwrotną.

PROGRAMOWANIE PROCESORA 6502

Procesor 6502 dzięki swej nieskomplikowanej architekturze i niedużej liczbie bardzo efektywnych rozkazów jest przyjazny nawet dla mniej wprawnych programistów. Pomimo stosunkowo wolnego "zegara", czyli generatora impulsów taktujących, zastosowanego w ośmiobitowych komputerach ATARI, programy wykonują się szybko, a przy tym zajmują niewiele pamięci.

1. Pamięć, adresy, rejestry

Procesor 6502 może odwoływać się do 65536 komórek pamięci zwanych bajtami. Jeden bajt jest fragmentem pamięci umożliwiającym przechowanie pojedynczego znaku. Znak w tym rozumieniu utożsamiany jest ze swym numerem (kodem) w zestawie ATASCII, czyli liczbą od 0 do 255. Z punktu widzenia procesora nie jest możliwy podział pamięci na mniejsze elementy. Dwa kolejne bajty nazywamy słowem. Jako adres słowa przyjmuje się adres jego pierwszego bajtu. Zawartość słowa może być rozumiana jako liczba od 0 do 65535, co umożliwia ponumerowanie wszystkich bajtów w pamięci. Taki jednoznaczny numer nazywa się adresem. Dla wykonania operacji na konkretnym bajcie należy sprecyzować jego adres. Ponieważ dla określenia adresu potrzebne są dwa bajty, mówimy o starszym (bardziej znaczącym) i młodszym (mniej znaczącym) bajcie. Dla słów przyjęto zasadę, w myśl której młodszy bajt zajmuje niższą lokację (ma mniejszy adres) niż starszy. Grupę bajtów pamięci o jednakowej starszej połówce adresu nazywamy stroną. Stronę, której starsza połowa adresu (numer) jest równa 0, zwie się zerową.

Trzy z rejestrów procesora są bezpośrednio dostępne dla programisty. Jest to rejestr danych zwany A (akumulator), oraz dwa rejestry indeksowe (X i Y). Pozostałe rejestry, do których dostęp jest nieco utrudniony to: wskaźnik instrukcji PC, wskaźnik stosu SP, rejestr znaczników F. Użycie tych rejestrów polega przeważnie na zmianach niejawnych, co będzie wyjaśnione przy omawianiu konkretnych rozkazów. Rejestry A, X, Y i F mają długość jednego bajtu każdy, PC jest dwubajtowy. Co do SP, to zdania są podzielone. Z punktu widzenia procesora jest dwubajtowy, lecz starszy bajt jest stale równy 1. Dla programisty jest to rejestr jednobajtowy, ponieważ modyfikacji podlega (jest dostępna) tylko młodsza jego połowa.

2. Budowa rozkazu

Rozkazem procesora jest kawałek pamięci zaadresowany przez aktualną zawartość rejestru PC. W zależności od pierwszego ze znalezionych tam bajtów w skład rozkazów mogą wchodzić jeszcze kolejne bajty. W sumie rozkaz może mieć długość od jednego do trzech bajtów. Nie wszystkie możliwe wartości zawarte w pierwszym bajcie, zwanym kodem rozkazu, są prawidłowe. Dla nieprawidłowych (nielegalnych) kodów reakcja procesora, czyli interpretacja rozkazu, może być (i na ogół jest) nieprzewidziana. Prawidłowe rozkazy dzielimy najprościej ze względu na długość. Do rozkazów jednobajtowych należą rozkazy wewnętrzne procesora (przełączanie między rejestrami, manipulacja znacznikami) i większość operacji na stosie. Rozkazy dwubajtowe to: skoki względne, operacje na stronie zerowej, rozkazy o adresacji pośredniej z indeksem oraz rozkazy z argumentem "natychmiastowym", czyli zawartym w samym rozkazie. Trzabajtowe rozkazy to operacje na pamięci poza stroną zerową i różne skoki.

3. Tryby adresacji

Celem adresowania pamięci jest odszukanie argumentu dla rozkazu procesora. Na następnej stronie widnieje opis trybów adresowania, w którym:

- a oznacza liczbę z zakresu 0..65535
- b oznacza liczbę z zakresu 0..255
- (X), (Y) oznacza zawartość odpowiedniego rejestru
- [n] oznacza zawartość słowa pamięci o adresie n
- (n) oznacza zawartość bajtu pamięci o adresie n

Uwaga: W przypadku adresacji pośredniej indeksowej adres pośredni musi znajdować się na stronie zerowej.

| | <u>adresacja</u> | <u>zapis</u> | <u>argument</u> |
|-----|---------------------|--------------|-----------------|
| 1. | natychmiastowa | #b | |
| 2. | bezwzględna długa | a | (a) |
| 3. | bezwzględna krótka | b | (b) |
| 4. | indeksowa długa | a,X | (a + (X)) |
| 5. | | a,Y | (a + (Y)) |
| 6. | indeksowa krótka | b,X | (b + (X)) |
| 7. | | b,Y | (b + (Y)) |
| 8. | pośrednia indeksowa | (b,X) | ([b + (X)]) |
| 9. | | (b),Y | ([b] + (Y)) |
| 10. | pośrednia | (a) | ([a]) |

4. Strona zerowa

Szczególne znaczenie strony zerowej polega na możliwości odwołań do niej przez podanie tylko połowy (młodsze bajtu) adresu. Dzięki temu rozkazy wykonujące operacje na stronie zerowej są krótsze i szybciej działają. Niestety, wypada też spostrzec, że strona zerowa nie jest z gumy. Poza tym rozkazy działające na stronie zerowej trzeba stosować umiejętnie, pamiętając, że starszy bajt wyliczonego adresu efektywnego jest zawsze zerem. Zatem np. zapis postaci 255,X, gdy rejestr X zawiera 5, odwoła się faktycznie do bajtu o adresie 4, nie zaś 260, jak się czasem sądzi. Podobnie odwołanie (255),Y użyje adresu pośredniego, którego młodszy bajt znajduje się pod adresem 255, a starszy pod adresem 0!

5. Stos

Stosem nazywamy pierwszą stronę pamięci adresowaną poprzez specjalny rejestr zwany wskaźnikiem stosu (SP). Starszy bajt wskaźnika, jako numer strony, jest zawsze równy 1. Młodszy bajt wskazuje miejsce na stosie, do którego aktualnie można się odwołać. Operacja umieszczenia bajtu na stosie polega na zapisaniu jego wartości w miejscu wskazanym przez wskaźnik stosu, po czym młodszy bajt wskaźnika stosu ulega zmniejszeniu o jeden. Wskaźnik pokazuje zawsze pierwszy wolny bajt na stosie. Operacja

zdejmowania ze stosu przebiega odwrotnie. Najpierw zwiększany jest wskaźnik stosu, następnie pobiera się wartość przezeń wskazaną. Stąd wniosek: wartość młodszego bajtu wskaźnika: 255 oznacza stos pusty lub pełny zależnie od ostatnio wykonanej na nim operacji. Trzeba bowiem pamiętać, że zmniejszenie o 1 bajtu o wartości 0 daje w rezultacie 255!

6. Znaczniki

Rejestr znaczników stanowi zespół ośmiu jednobitowych przełączników ustawianych przez procesor w zależności od wyniku ostatnio wykonanej operacji. Są instrukcje, które w ogóle nie wpływają na stan znaczników. Inne zmieniają tylko niektóre z nich. Poniższa tabelka opisuje pozycję w bajcie, literowy kryptonim i objaśnienie każdego znacznika.

| nr bitu | znacznik | 1 oznacza... |
|---------|----------|----------------------------------------|
| 7 | N | 7 bit wyniku = 1 |
| 6 | V | przeniesienie (lub pożyczkę) z 6 bitu |
| 5 | - | nie używany, zawsze 1 |
| 4 | B | poza obsługą przerwania sprzętowego |
| 3 | D | ustawiony tryb dziesiętny |
| 2 | I | obsługa przerw IRQ jest zabroniona |
| 1 | Z | wynik = 0 |
| 0 | C | przeniesienie (brak pożyczki) z 7 bitu |

Pewnego komentarza wymaga znacznik B. Pozwala on odróżnić przerwanie sprzętowe (wywołane przez sygnał na nóżce procesora) od przerwania programowego (rozkaz BRK) po tym, że w przypadku przerwania sprzętowego znacznik ten ustawia się na 0.

7. Rozkazy procesora 6502

Naturalnym wydaje się raczej podział rozkazów nie ze względu na długość, lecz na realizowane funkcje. Tak więc wydzielimy tu 3 grupy rozkazów (wszystkie kody podane są w notacji szesnastkowej):

7.1. Rozkazy przesłań do i z pamięci, arytmetyczne i logiczne

| nazwa | działanie | tryby adresacji | | | | | | | | | znaczniki | | |
|-------|---------------------------------|-----------------|----|----|----|----|----|----|----|----|-----------|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | N | V | Z |
| LDA | $A \leftarrow M$ | A9 | AD | A5 | BD | B9 | B5 | | A1 | B1 | x | . | x |
| LDX | $X \leftarrow M$ | A2 | AE | A6 | | BE | | B6 | | | x | . | x |
| LDY | $Y \leftarrow M$ | A0 | AC | A4 | BC | | B4 | | | | x | . | x |
| ADC | $A \leftarrow A+M+C$ | 69 | 6D | 65 | 7D | 79 | 75 | | 61 | 71 | x | x | x |
| SBC | $A \leftarrow A-M-1+C$ | E9 | ED | E5 | FD | F9 | F5 | | E1 | F1 | x | x | x |
| CMP | $A-M$ | C9 | CD | C5 | DD | D9 | D5 | | C1 | D1 | x | . | x |
| CPX | $X-M$ | E0 | EC | E4 | | | | | | | x | . | x |
| CPY | $Y-M$ | C0 | CC | C4 | | | | | | | x | . | x |
| AND | $A \leftarrow A \text{ and } M$ | 29 | 2D | 25 | 3D | 39 | 35 | | 21 | 31 | x | . | x |
| EOR | $A \leftarrow A \text{ eor } M$ | 49 | 4D | 45 | 5D | 59 | 55 | | 41 | 51 | x | . | x |
| ORA | $A \leftarrow A \text{ or } M$ | 09 | 0D | 05 | 1D | 19 | 15 | | 01 | 11 | x | . | x |
| BIT | *(patrz niżej) | | 2C | 24 | | | | | | | x | x | x |
| DEC | $M \leftarrow M-1$ | | CE | C6 | DE | | D6 | | | | x | . | x |
| INC | $M \leftarrow M+1$ | | EE | E6 | FE | | F6 | | | | x | . | x |
| STA | $M \leftarrow A$ | | 8D | 85 | 9D | 99 | 95 | | 81 | 91 | . | . | . |
| STX | $M \leftarrow X$ | | 8E | 86 | | | | 96 | | | . | . | . |
| STY | $M \leftarrow Y$ | | 8C | 84 | | | 94 | | | | . | . | . |

Rozkazy te komunikują się z pamięcią i dla każdego z nich dopuszcza się zastosowanie różnych trybów adresacji. W powyższej tabelce zestawiono kody tych rozkazów w zależności od trybu adresowania. Tryby ponumerowano tak, jak w punkcie 3. Rozkazy te oddziałują zwykle na bity rejestru znaczników. W kolumnie "znaczniki" x oznacza bit modyfikowany przez dany rozkaz, zaś . - pozostawiany bez zmian. Pod hasłem "działanie" M oznacza komórkę pamięci wyznaczoną zgodnie z zastosowanym trybem adresacji. C oznacza znacznik C (0 lub 1).

Rozkaz BIT ustawia trzy znaczniki: do N i V przenoszony jest odpowiednio 7 i 6 bit argumentu (bez względu na zawartość akumulatora), znacznik Z ustawiany jest tak, jak po wykonaniu operacji "and" argumentu z akumulatorem, wynik tej operacji nie jest wszakże nigdzie przesyłany.

Egzotyczny symbol "@", określający jeden z trybów adresacji następnych czterech rozkazów, oznacza tzw. adresowanie akumulatora. Rozkazy te mogą być bowiem jedno-, dwu-, lub trzybajtowe i w swej jednobajtowej odmianie stają się rozkazami wewnętrznymi procesora operującymi na zawartości akumulatora. Rozkazy ASL i ROL powodują przesunięcie bitów argumentu (lub akumulatora) o jedną pozycję w lewo. Bit 6 znajdzie się zatem na miejscu 7, 5 na 6, itd. Bit 7 przenoszony jest do znacznika C. Bit 0 jest w przypadku ASL zerowany, a przy ROL - przenoszony ze znacznika C. Rozkazy LSR i ROR powodują analogiczne przesunięcie bitów w prawo. LSR zeruje najstarszy bit, a ROR przenosi go z C. Najmłodszy bit przechodzi do C.

| nazwa | ----- tryby adresacji ----- | | | | | | znaczniki | | |
|-------|-----------------------------|----|----|----|---|-------|-----------|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 @ | N | Z | C |
| ASL | | 0E | 06 | 1E | | 16 0A | x | x | x |
| ROL | | 2E | 26 | 3E | | 36 2A | x | x | x |
| LSR | | 4E | 46 | 5E | | 56 4A | x | x | x |
| ROR | | 6E | 66 | 7E | | 76 6A | x | x | x |

7.2. Rozkazy operacji na stosie i/lub wskaźniku instrukcji

Rozkazy warunkowych skoków względnych Bxx są dwubajtowe, lecz nie mówi się tu o trybie adresacji, gdyż drugi bajt instrukcji wykorzystywany jest dla obliczenia adresu skoku. Bajt ten traktowany jest jako liczba ze znakiem (od -128 do 127) i dodaje się go (oczywiście w przypadku spełnionego warunku) do wskaźnika instrukcji, co daje skoki w przód lub w tył w tym zakresie. Adres skoku liczy się od miejsca po instrukcji skoku. Jest to stan wskaźnika instrukcji, który utrzyma się, gdy warunek skoku nie jest spełniony.

| <u>nazwa</u> | <u>kod</u> | <u>skok nastąpi gdy znacznik...</u> |
|--------------|------------|-------------------------------------|
| BPL | 10 | N = 0 |
| BMI | 30 | N = 1 |
| BVC | 50 | V = 0 |
| BVS | 70 | V = 1 |
| BCC | 90 | C = 0 |
| BCS | B0 | C = 1 |
| BNE | D0 | Z = 0 |
| BEQ | F0 | Z = 1 |

Rozkazy skoków bezwarunkowych są zawsze trzybajtowe. Rozkazy 4C i 20 zawierają adres skoku, natomiast rozkaz 6C zawiera adres słowa zawierającego adres skoku. Jest to zatem skok pośredni. Rozkazy JMP modyfikują tylko wskaźnik instrukcji, zaś JSR zapisuje dodatkowo na stosie adres "powrotu". Jest to konkretnie adres o 1 mniejszy od adresu następnej po JSR instrukcji, co umożliwia powrót do porzuconej JSR-em sekwencji rozkazów za pomocą instrukcji RTS. Na stosie umieszczany jest najpierw starszy bajt adresu powrotu, potem młodszy. Rozkaz RTS zdejmując ze stosu dwa bajty i tak otrzymany adres po zwiększeniu o jeden wstawiony zostaje do wskaźnika instrukcji.

| <u>nazwa</u> | <u>kod</u> |
|--------------|------------|
| JMP | 4C |
| JMP | 6C |
| JSR | 20 |
| RTS | 60 |

Rozkaz BRK wywołuje przerwanie programowe, pod względem przebiegu nie odbiegające od przerwania sprzętowych. Na stosie zostają umieszczone trzy bajty: adres powrotu z przerwania (2) oraz bieżący stan rejestru znaczników (1). Rozkaz RTI wykonuje powrót z przerwania zdejmując te bajty w odwrotnej kolejności. Pozostałe rozkazy operujące na stosie to: PHP umieszczający na stosie bajt znaczników, PLP zdejmujący ten bajt, oraz PHA i PLA, odpowiednio, kładący na stos zawartość akumulatora i zdejmujący ją.

| <u>nazwa</u> | <u>kod</u> |
|--------------|------------|
| BRK | 00 |
| RTI | 40 |
| PHP | 08 |
| PLP | 28 |
| PHA | 48 |
| PLA | 68 |

Uwaga: rozkazy RTI, PLP, PLA zmieniają rejestr znaczników.

7.3. Rozkazy wewnętrzne procesora

Rozkazy modyfikujące wybrane bity rejestru znaczników zgrupowane są w poniższej tabelce:

| nazwa | kod | ---- znaczniki --- | | | | | |
|-------|-----|--------------------|---|---|---|---|---|
| | | N | V | D | I | Z | C |
| CLC | 18 | . | . | . | . | . | 0 |
| SEC | 38 | . | . | . | . | . | 1 |
| CLI | 58 | . | . | . | 0 | . | . |
| SEI | 78 | . | . | . | 1 | . | . |
| CLV | B8 | . | 0 | . | . | . | . |
| CLD | D8 | . | . | 0 | . | . | . |
| SED | F8 | . | . | 1 | . | . | . |

Ostatnia tabela zawiera rozkazy przesłań między rejestrami procesora, zwiększania i zmniejszania rejestrów indeksowych, oraz rozkaz NOP, który ze względu na podobieństwo kodu wydaje się tu pasować. Rozkaz NOP nic nigdzie nie przesyła ani nic nie ustawia.

| nazwa | działanie | kod | znaczniki | | | |
|-------|--------------------|-----|-----------|---|---|---|
| | | | N | V | Z | C |
| TXA | $A \leftarrow X$ | 8A | x | . | x | . |
| TXS | $SP \leftarrow X$ | 9A | . | . | . | . |
| TAX | $X \leftarrow A$ | AA | x | . | x | . |
| TSX | $X \leftarrow SP$ | BA | x | . | x | . |
| DEX | $X \leftarrow X-1$ | CA | x | . | x | . |
| DEY | $Y \leftarrow Y-1$ | 88 | x | . | x | . |
| TYA | $A \leftarrow Y$ | 98 | x | . | x | . |
| TAY | $Y \leftarrow A$ | A8 | x | . | x | . |
| INY | $Y \leftarrow Y+1$ | C8 | x | . | x | . |
| INX | $X \leftarrow X+1$ | E8 | x | . | x | . |
| NOP | | EA | . | . | . | . |

JĘZYK ASEMBLERA

Słowo "assembler" oznacza rodzaj programu tłumaczącego teksty napisane przy użyciu pewnego formalizmu, zwanego językiem symbolicznym (lub językiem assemblera), na liczby, będące programem konkretnego procesora i zdadne do umieszczenia w pamięci komputera. Dość rozpowszechniony zwyczaj nazywania tego formalizmu również "assemblerem" stanowi przyczynę terminologicznego zamętu. Język assemblera jest przedstawieniem języka maszynowego (będącego ciągiem liczb) w postaci czytelnej i zrozumiałej dla człowieka, poprzez zastąpienie kodów rozkazów ich nazwami. Również odwołania do miejsc pamięci mogą być zapisane w sposób symboliczny. Warto zwrócić uwagę, że o ile przedstawione w poprzednim rozdziale cechy języka (zbioru rozkazów) procesora 6502 stanowią reguły stałe, o tyle różne assemblyery mogą się różnić w szczegółach. Dotyczy to zwłaszcza grupy tzw. pseudorozkazów. Poniższy opis dotyczy, rzecz jasna, assemblera QA.

1. Sposób pracy assemblera, licznik lokacji

Assembler jest dwuprzebiegowy, to znaczy, że dla pełnego procesu tłumaczenia musi odczytać tekst programu dwukrotnie. W pierwszym przebiegu tworzona jest tablica etykiet oraz wykrywane są niektóre błędy. W drugim przebiegu powstaje kod wynikowy, wyświetlany i/lub drukowany może być listing programu oraz komunikaty o błędach. Kod programu (program maszynowy) może być umieszczany w pamięci lub zapisany w pliku na urządzeniu zewnętrznym. W każdym przypadku assembler planuje umieszczanie kolejnych bajtów kodu w kolejnych komórkach pamięci. Licznikiem lokacji nazywamy adres wskazujący na miejsce w pamięci, gdzie ma być umieszczony następny bajt kodu.

2. Liczby, etykiety, wyrażenia

Liczba może być przedstawiona jako:

- ciąg cyfr dziesiętnych np. 23456
- ciąg cyfr dwójkowych poprzedzonych znakiem % np. %101010011
- ciąg cyfr szesnastkowych ze znakiem \$ np. \$2FC

- pojedynczy znak ATASCII ujęty w apostrofy np. 'h'
- znak z gwiazdką oznaczającą +128 (negatyw) np. '?'

Etykieta jest symbolem reprezentującym pewną wartość liczbową. Etykietę stanowi ciąg od 1 do 6 liter, cyfr i znaków podkreślenia, z tym, że pierwszy znak nie może być cyfrą. Nie jest zalecane także używanie znaku podkreślenia na początku nazwy, ponieważ taką konwencję stosują firmowe podprogramy standardowe, unikniesz w ten sposób ewentualnego konfliktu nazw. Można używać etykiet dłuższych niż 6-znakowe, lecz assembler rozpoznaje tylko 6 początkowych znaków. Istnieją dwa sposoby definiowania etykiet. Pierwszy polega na oznaczeniu etykietą dowolnej instrukcji (lub pseudoinstrukcji). Wówczas etykieta przyjmuje wartość licznika lokacji w oznaczonym miejscu, inaczej mówiąc jest to adres, pod jakim będzie umiejscowiony oznaczony rozkaz (lub dana). Drugi sposób wiąże się z użyciem pseudorozkazu EQU i będzie omówiony dalej. W programie można użyć co najwyżej 512 etykiet.

Wyrażenie to ciąg liczb i/lub etykiet połączonych znakami + i - . W wyrażeniach można też używać na prawach liczby specjalnego symbolu * , oznaczającego aktualny (przed przetłumaczeniem danej instrukcji) licznik lokacji.

3. Format rozkazu

Każdy wiersz programu musi spełniać pewne wymagania formalne. Długość wiersza nie może przekraczać 64 znaków. Wiersz pusty (złożony z samych spacji) oraz wiersz, którego pierwszym znakiem jest * (lub dowolny znak semigraficzny), są pomijane. Służą one wymogom czytelności programu, dla wprowadzania odstępów i komentarzy. Pozostałe wiersze (rozkazy) dzielą się na pewną liczbę tzw. pól. Pierwszym polem jest pole etykiety. Zaczyna się ono od pierwszej kolumny. Jeśli dana instrukcja nie ma etykiety, pierwszą kolumnę należy pozostawić wolną. Drugie pole jest polem nazwy rozkazu (mnemonika). Trzyliterowe nazwy rozkazów wymienione są w rozdziale o 6502. Następnym polem jest pole argumentu. Rozkazy bezargumentowe nie mają tego pola. Jeśli jest, format argumentu zależy od użytego trybu adresacji. W miejscu przeznaczonym na liczbę może wystąpić dowolne wyrażenie. Ostatnim polem jest pole komentarza. Można w nim umieszczać dowolne znaki. Długość pól nie jest stała, a ich granice rozpozna-

wane są przez asembler po co najmniej jednej spacji. Wewnątrz pola, z wyjątkiem komentarza, spacje są zabronione.

Choć długości pól nie są sztywno narzucone, istnieje tendencja do lokowania ich w stałych kolumnach, co stwarza pożądany ład wzrokowy. Edytor QA pozwala stosować klawisz TAB, który powoduje przeskoczenie do początku najbliższego pola. Postulowany podział przewiduje zastosowanie czteroznakowych etykiet, co ułatwia zmieszczenie tekstu programu w 20 kolumnach (tyle tekstu źródłowego pokazuje "listing" wyświetlany podczas asemblacji). Zwolennicy innego rozplanowania pól mogą bez trudu przeredagować mapę tabulacji. Przykłady poprawnie zbudowanych wierszy programu:

| | | | | |
|------|-----|-----|--------|-------|
| | TXA | | | |
| ETYK | ASL | 10 | | |
| ROR | ROR | ROR | ROR | |
| SBC | | #64 | czemu? | |
| * | LDZ | PCW | | |
| | | LDA | | 44444 |

Przykłady najczęstszych błędów:

LAB LDX #10

(spacja przed etykietą)

ORA#27

(brak odstępu przed argumentem)

LABEL35 LDX #12
 LABEL37 LDY #13

(powtórna deklaracja etykiety LABEL3)

4. Pseudorozkazy, definiowanie danych

Pseudorozkazami nazywamy takie instrukcje asemblera, które nie są tłumaczone na język maszynowy. Służą do organizacji procesu asemblowania, definiowania etykiet, itd. Format tych instrukcji jest taki sam, jak pozostałych: opcjonalna etykieta, mnemonik, argument, komentarz.

4.1. OPT

Pseudorozkaz OPT służy do określenia parametrów asemblacji. Argumentem jest liczba z zakresu 0..127, której bity wpływają na konkretne opcje asemblera.

| --bity argumentu-- | | | | | | | znaczenie |
|--------------------|---|---|---|---|---|---|-------------------------------------------------|
| 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| . | . | . | . | . | x | x | poziom listowania |
| | | | | | 0 | 0 | – bez listingu |
| | | | | | 0 | 1 | – tylko rozkazy błędne |
| | | | | | 1 | 0 | – pełny listing programu z edytora |
| | | | | | 1 | 1 | – jak wyżej, wraz z dołączonymi plikami |
| . | . | . | . | x | . | . | wyprowadzenie listingu na ekran |
| . | . | . | x | . | . | . | wyprowadzenie listingu na drukarkę |
| . | . | x | . | . | . | . | umieszczenie kodu w pamięci komputera |
| x | x | . | . | . | . | . | produkcja kodu wynikowego do pliku |
| 0 | 0 | | | | | | – bez kodu |
| 0 | 1 | | | | | | – postać ładowalna DOS-em (z nagłówkami bloków) |
| 1 | 0 | | | | | | – jak wyżej, dołączenie do istniejącego pliku |
| 1 | 1 | | | | | | – sam kod programu, bez nagłówków |

Bit 7 musi być zerem. Rozkazu tego można używać w programie wielokrotnie, tłumacząc różne partie programu w różnych trybach. Aseblowanie na urządzenie zewnętrzne wszakże powinno obejmować

całość programu albo kompletny blok, wtedy należy je włączyć bezpośrednio po pseudorozkazie ORG, i wyłączyć w takim samym miejscu. W przeciwnym razie zapisany kod może okazać się nieprawidłowy. Przed pierwszym użyciem rozkazu OPT asembler jest ustawiony tak, jak po OPT 6. Przykłady:

```
OPT    %10101
```

(kod do pamięci, listowanie błędów na ekranie)

```
OPT    10
```

(wydruk listingu)

4.2. ORG

Pseudorozkaz ORG służy do ustawienia nowej wartości licznika lokacji. Przed pierwszym użyciem tej instrukcji licznik lokacji ma wartość 0. W programie można użyć tego rozkazu co najwyżej 63 razy. Uwaga: etykieta nadana temu rozkazowi otrzyma wartość dotychczasowego licznika. Używany bywa w dwóch celach: do określenia początkowego adresu bloku instrukcji programu lub danych, np.

```
ORG    $600
```

oraz do definiowania obszarów roboczych w pamięci, np.

```
BYT1    ORG    *+1  
BYT2    ORG    *+1  
WORD    ORG    *+2
```

itd.

4.3. EQU

Pseudorozkaz EQU służy do nadawania wartości etykietie (definiowania etykiety). Etykieta może być zdefiniowana tylko raz. Wszystkie elementy argumentu muszą być zdefiniowane w programie przed użyciem tej instrukcji. Poprzedzenie rozkazu EQU etykietą jest obowiązkowe. Przykłady:

| | | |
|------|-----|--------|
| Z | EQU | 5 |
| DWAZ | EQU | Z+Z |
| Z_3 | EQU | Z+DWAZ |
| IOCB | EQU | \$340 |

4.4. END

Bezargumentowy pseudorozkaz END wymusza na assemblerze zakończenie analizy tekstu programu (lub pliku dołączonego). Ładnie napisany program zawsze kończy się tym rozkazem, po czym poznać, że programista zakończył pracę, a nie zasnął w jej trakcie.

4.5. ICL

Pseudorozkaz ICL powoduje dołączenie, w miejscu jego wystąpienia, pliku z dodatkową częścią programu. W skrajnym przypadku edytor może zawierać tylko tę jedną instrukcję. Argumentem rozkazu jest nazwa pliku dodatkowego ujęta w apostrofy. Po zakończeniu asemblacji pliku dodatkowego assembler powróci do przetwarzania tekstu w edytorze. Pseudoinstrukcje OPT w pliku dodatkowym są ignorowane. Możliwe jest łańcuchowanie plików poprzez użycie w pliku dołączanym kolejnej instrukcji ICL. Powinna to być ostatnia instrukcja, ponieważ po napotkaniu instrukcji END lub końca pliku dołączanego, assembler powraca zawsze do programu w edytorze. Oczywiście z uwagi na dwa przebiegi cała ta operacja powtarza się dwukrotnie. Przykład:

ORG \$600
ICL 'D8:PG6.TXT'

W przypadku braku rozszerzenia dodawane jest automatycznie .ASM . Jeśli pominąć nazwę urządzenia, QA odwoła się do tej stacji dysków, z której został wczytany (gdy QA nie był wczytywany z dysku, to nazwę urządzenia trzeba podać jawnie). W razie zastosowania pamięci taśmowej, ze względu na dwa przebiegi asemblacji, trzeba dwukrotnie ustawiać do wczytywania dołączane pliki. Dlatego w takim przypadku lepiej (o ile się zmieści) scalić kawałki programu rozkazem **^Kr** edytora.

4.6. DTA

DTA to osobny typ instrukcji, ponieważ powoduje wygenerowanie konkretnych danych. Dane te mogą być jednego z sześciu typów:

- A(adres) – produkuje słowo wypełniając je podaną wartością
- B(bajt) – produkuje bajt wypełniając go podaną wartością
- C'tekst' – produkuje ciąg kodów ATASCII podanych znaków
- D'tekst' – produkuje ciąg kodów ekranowych podanych znaków
- L(adres) – produkuje młodszy bajt z podanej wartości
- H(adres) – produkuje starszy bajt z podanej wartości

"adres", "bajt" to wyrażenia o wartościach odpowiednio 0..65535 i 0..255. "tekst" to dowolny ciąg znaków (apostrof w takim tekście przedstawia się jako parę apostrofów). Dodatkowa opcja: * po kończącym apostrofie powoduje wygenerowanie podanego tekstu w negatywie. Ta opcja stanowi ukłon (zachowanie zgodności) w stronę JBW Assemblera, którego edytor nie dopuszczał tekstów w negatywie. W QA takie ograniczenie nie występuje. Można więc w tym znaczeniu gwiazdki nie używać. Adresem, od którego assembler umieszcza dane, jest aktualny licznik lokacji. Pojedyncza instrukcja DTA może zawierać kilka

danych, rozdzielamy je wtedy przecinkami. Pamiętać należy przy tym, że symbol *, oznaczający licznik lokacji, zmienia wartość po każdej takiej danej.

5. Różne drobne i grube uwagi

Pewne tryby adresacji wymagają podania liczby jednobajtowej. Asembler wymaga aby wszystkie składniki wyrażenia opisującego ten argument były zdefiniowane przed użyciem. Podobny wymóg dotyczy pseudorozkazu EQU.

Adresację natychmiastową w stosunku do standardowego minimum wzbogacono o dwa tryby: użycie znaku < zamiast # powoduje wzięcie młodszego bajtu argumentu, a znaku > – starszego. Sposób ten pozwala np. obejść wymóg deklarowania argumentów natychmiastowych przed ich użyciem. Nowe tryby pozwalają bowiem na stosowanie dowolnych liczb, w tym również deklarowanych w dalszej części programu.

Rozkazy ASL, ROL, LSR, ROR mogą, jak wiadomo, występować z adresem lub bez. W tym drugim przypadku należy wpisać w miejscu argumentu znak @.

Rozkazy warunkowych skoków względnych są traktowane przez asembler w sposób szczególny. Ich argumentami bowiem są w języku asemblera (w przeciwieństwie do maszynowego) rzeczywiste adresy skoków (bezwzględne) lub, rzecz jasna, stosowne wyrażenia. Asembler przelicza je na liczby względne (odległości) i tak umieszcza w rozkazach. Warto spostrzec, że BCC *+10 tłumaczy się na maszynowe BCC +8 ze względu na różny punkt odniesienia.

Posiadaczy JBW Assembler-a ucieszy zapewne informacja o 100% zgodności QA z tamtym asemblerem. Oznacza to, że każdy program napisany dla asemblera "JBW" będzie działał bez zmian w systemie QA. Jedynym wymogiem jest zmiana formatu tekstu ze specjalnego (upakowanego) formatu "JBW" na plik ASCII, akceptowany przez QA. Do tego celu służy program TRANS.COM, którego wersja źródłowa TRANS.ASM została dołączona do zestawu. Nie każdy natomiast program, napisany dla wyposażonego

w szersze możliwości QA, spodoba się assemblerowi "JBW". Dlatego nie publikujemy konwertera w drugą stronę, każdy zresztą bez trudu może napisać go sobie sam.

6. Błędy asemblacji

Assembler reaguje na wykrycie błędu w tłumaczonym tekście wyświetleniem błędnego rozkazu, a miejsce, gdzie wykrył błąd, wyróżnia znakiem w negatywie (tylko do 20. kolumny). Pod błędną instrukcją pojawia się skrótowy komunikat. Oczywiście pokazanie tego musi być zezwolone odpowiednimi bitami rozkazu OPT. Oto wykaz komunikatów wraz z objaśnieniami:

| | |
|-------------------|-------------------------------------------------------------------------------------------------------|
| TOO BIG NUMBER | – użyto liczby większej od 65535 |
| DIGIT EXPECTED | – po \$ lub % wystąpił znak inny niż cyfra |
| STRING ERROR | – brak apostrofu zamykającego napis |
| ILLEGAL SYMBOL | – wystąpił symbol niedozwolony w tym miejscu |
| LABEL DECL TWICE | – powtórna deklaracja etykiety |
| TOO MANY LABELS | – liczba etykiet przekroczyła 512 |
| UNEXPECTED EOL | – wiersz niedokończony |
| UNBAL PARENTH | – brak nawiasu otwierającego lub zamykającego |
| UNDEFINED OR LONG | – użyto argumentu większego od 255 lub dotąd niezdefiniowanego w miejscu wymagającym liczby bajetowej |
| UNDECLARED LABEL | – użyto etykiety niezdefiniowanej w programie |
| BRANCH TOO FAR | – adres skoku względnego przekracza dopuszczalny zakres |
| IMPROPER TYPE | – nieodpowiedni typ argumentu |
| UNDEFINED VALUE | – argument dotychczas niezdefiniowany |
| LABEL MISSING | – brak etykiety przed EQU |
| MEMORY CONFLICT | – asemblacja w niedozwolony obszar pamięci |
| TOO MANY ORGS | – przekroczono limit instrukcji ORG |

QUICK ASSEMBLER

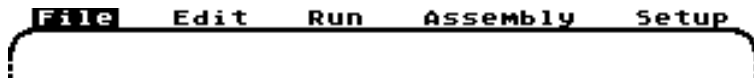
Quick Assembler (w skrócie QA) jest zintegrowanym pakietem edytora, asemblera i mini-debugera. Pozwala wyjątkowo szybko uruchamiać programy poprzez możliwość ich redagowania, asembrowania i testowania bez korzystania z pamięci zewnętrznej.

D Aby uruchomić QA z oryginalnej dyskietki, należy po wyświetlonej przez DOS zachęcie "D1:" napisać
QA
i nacisnąć RETURN

C Aby uruchomić QA z oryginalnej kasety, należy ustawić ją w miejscu, gdzie znajduje się ten program, po wyświetlonej przez COS zachęcie ">" napisać
**
i nacisnąć RETURN

Uwaga: systemowy wskaźnik MEMLO (zawarty w słowie \$2E7, a możliwy do odczytania w DOS-ie lub COS-ie rozkazem MEM) nie może przekraczać \$4800! Wyklucza to, niestety (z wyjątkiem nielicznych DOS-ów) obecność w pamięci opisanego w następnym rozdziale BH.

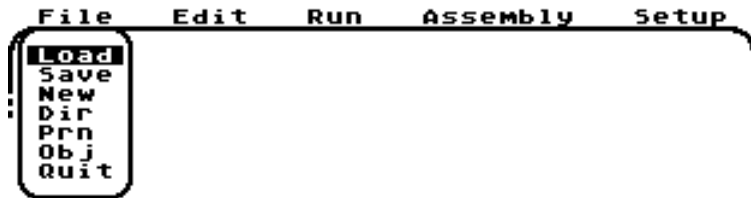
Zasady komunikowania się z użytkownikiem zbliżone są do programów firmy Borland znanych z większych komputerów. Górny wiersz ekranu zawiera podstawowe (główne) "menu":



Jedna z opcji wyróżniona jest podświetleniem, które można przemieszczać w prawo lub w lewo, naciśnięcie zaś RETURN powoduje wywołanie podświetlonej funkcji. Innym, nie gorszym, sposobem aktywacji opcji jest użycie pierwszej litery jej nazwy (nie trzeba już wówczas używać RETURN). Powrót do głównego menu umożliwia klawisz ESC.

1. File

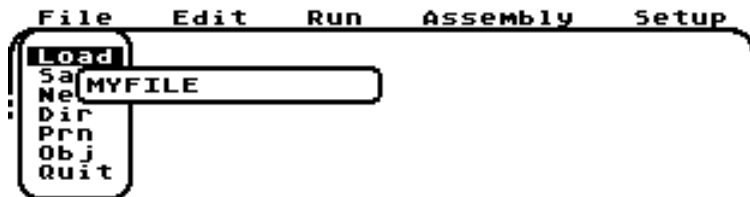
Pod hasłem tym zgrupowane są operacje dotyczące wejścia/wyjścia, w tym także wyjście do DOS-u (opuszczenie QA). Po wybraniu **File** otwiera się dodatkowe okno z opcjami:



Można je aktywizować przez wskazanie ruchomym podświetleniem i naciśnięcie RETURN, bądź bezpośrednio za pomocą pierwszej litery.

1.1. Load i Save

Load umożliwia wczytanie tekstu programu z nośnika pamięci zewnętrznej, a **Save** pozwala zapisać tekst na takim nośniku. Poprzedzone jest to zapytaniem o nazwę pliku, którą wpisuje się w specjalnie do tego celu otwartym okienku, np.:



Jeżeli się poda samą tylko nazwę, nie poprzedzoną określeniem urządzenia, to QA odwoła się do stacji dysków, z której został wczytany. Jeśli wczytano QA z innego urządzenia, lub, co gorsza, w ogóle nie dysponujemy napędem dysków, to trzeba określić jakieś "C:", czy "T:", z którym chce się współpracować.

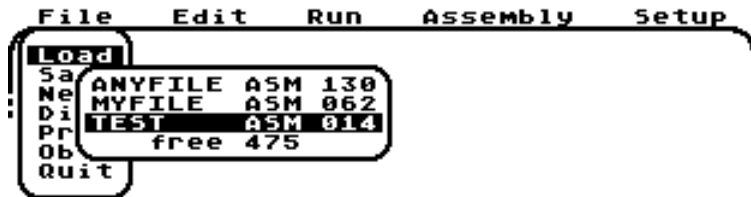
Okienko otwiera się tuż pod wybraną opcją (to ważne, bo żadne dodatkowe opisy czy zachęty się nie pojawiają) i zawiera "domyślną" (poprzednio użytą) nazwę, co pozwala na szybkie jej zaakceptowanie, jeśli się nie zmieniła. Gdy rozpocznie się wpisywanie nowej nazwy, stara w całości znika (nie ma potrzeby jej wymazywania), jeśli zaś użyć klawiszy edycyjnych, np. BACKSPACE, można dotychczasową nazwę poprawiać. Wprowadzanie nazwy kończy się klawiszem RETURN. Klawisz ESC natomiast pozwala przerwać operację (przywracana jest dotychczasowa nazwa pliku).

C Zapis i odczyt pliku na taśmie przebiega standardowo w trybie z krótkimi przerwami międzyrekordowymi. Aby wymusić długie przerwy, trzeba przy końcowym RETURN, akceptującym nazwę pliku, wcisnąć dodatkowo klawisz SHIFT. Długie przerwy są niezbędne, jeśli plik ma być asemblowany bezpośrednio z taśmy (przy użyciu pseudorozkazu ICL 'C:').

D Jeśli **Save** prowadzi do wymazania istniejącego już pliku o takiej samej nazwie, QA ostrzega o tym i pozwala w porę zrezygnować z zamiaru.

W przypadku, gdy nazwa zawiera znaki "*" lub "?" wyświetlone zostanie kolejne menu - spis plików, których nazwy odpowiadają podanemu szablónowi (np. "*.ASM"):

Można wówczas, używając ruchomego podświetlenia, wybrać plik z przedstawionej listy i zaakceptować go przez RETURN.



1.2. New

Funkcji tej używa się dla oczyszczenia bufora edytora ze znajdującego się tam dotąd tekstu. QA inicjuje przy tym nazwę nowego (pustego) pliku na NONAME.ASM. Przy zapisie utworzonego potem tekstu dobrze jest zmienić tę nazwę na inną.

1.3. Dir

D | **Dir** pozwala na obejrzenie katalogu dyskiety. Podanie szablonu (w sposób analogiczny do wprowadzania nazwy pliku) umożliwia wyłowanie tylko określonej grupy plików, np. "QA.*". Domyślnym urządzeniem, którego dotyczy ta operacja jest stacja dysków, z której uruchomiono QA. W stosunku do innej stacji trzeba określić jej symbol, np. "D2:*.COM". Klawisz ESC kończy przeglądanie katalogu, jeśli zamiast niego użyjemy RETURN, to nazwa pliku wskazanego ruchomym podświetleniem zostanie przeniesiona do pola nazwy w operacjach **Load** i **Save**.

C | Tej funkcji się nie używa.

1.4. Prn i Obj

Opcje **Prn** i **Obj** służą do określenia aktualnych plików (urządzeń) wyjściowych dla asemblera, do których skieruje on odpowiednio wydruk i kod wynikowy. Redagowanie nazw identycznie, jak w przypadku **Load** i **Save**. Należy tylko pamiętać, że każde wprowadzenie nazwy dla **Load** lub **Save** powoduje przepisanie jej do pola **Obj** (bez rozszerzenia, które przyjmuje domyślną postać ".OBJ"). Dlatego, jeśli plik ów ma się inaczej nazywać, trzeba to poprawić p o zmianie nazwy dla **Load/Save**. Domyślnym rozszerzeniem dla wydruku jest oczywiście ".PRN".

C | W przypadku wysłania wydruku lub kodu na taśmę, QA stosuje tryb z krótkimi przerwami międzykordowymi. Aby wymusić długie przerwy, trzeba przy naciskaniu klawisza "A" lub RETURN, uruchamiającym asemblację, wcisnąć dodatkowo klawisz SHIFT.

Uwaga: nie należy kierować obu tych plików na "C:" jednocześnie!

1.5. Quit

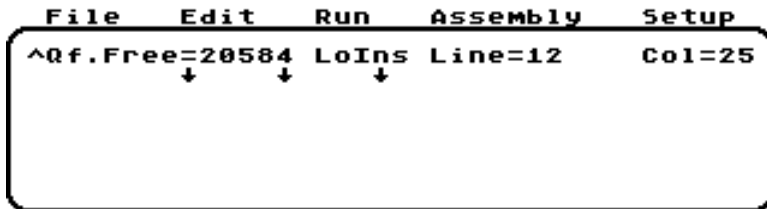
Quit powoduje zakończenie pracy QA i powrót do systemu. QA wzywa do zachowania redagowanego pliku, o ile nie zostało to wcześniej zrobione. Podobne przypomnienia pojawiają się także we wszelkich sytuacjach, które prowadzą lub choćby mogą prowadzić do utraty tekstu nie zachowanego programu. Pojawia się ostrzeżenie:



Ostrzeżenie to związane jest ze znacznikiem modyfikacji pliku (patrz: **Edit**). Można wybrać odpowiedź przez naprowadzenie na nią podświetlenia i naciśnięcie RETURN, albo klawiszem "S" lub "C". Wybór **Continue** oznacza rezygnację z zachowania pliku w obecnym jego kształcie, powodując skasowanie znacznika modyfikacji. **Save** wymaga dodatkowo podania nazwy pliku, tak samo, jak w głównym **Save**. Klawisz ESC pozwala wycofać się z niebezpiecznej operacji bez zapisywania pliku i bez kasowania wspomnianego znacznika.

2. Edit

Wyraz **Edit** oznacza redagowanie (tekstu). Ta opcja uaktywnia edytor tekstów, którego robocza płaszczyzna rozpościera się wewnątrz ramki tuż pod głównym menu:



2.1. Informacje o stanie edytora

Pierwszy wiersz opisuje tryb pracy i stan edytora, drugi przedstawia położenie przystanków tabulacji. Wiersz stanu jest niestety z uwagi na szczupłość miejsca znacznie zagęszczony. Oto kolejne elementy (od lewej do prawej): Symbol **^Qf** jest przykładem echa rozkazu specjalnego wydanego edytorowi w trybie "WordStar" i widnieje tylko przez chwilę, od wprowadzenia rozkazu do jego realizacji (przeważnie to miejsce jest puste).

Kropka przed Free oznacza, że plik zawarty w edytorze został zmodyfikowany od czasu ostatniego z nim pożegnania. Kropka ta (znacznik modyfikacji pliku) znika po prawidłowym wykonaniu operacji **Load**, **Save** lub **New**, a także na skutek rezygnacji z zapisania pliku po ostrzeżeniu "File modified!". Pojawia się natomiast pod wpływem zmian dokonywanych w redagowanym pliku. Uwaga: niekiedy zmian dokonuje sam edytor, przeważnie gdy format wczytanego tekstu nie jest odpowiedni (np. brak E-O-L na końcu).

Free=20584 przedstawia wielkość pamięci pozostałej do wykorzystania. Nie należy dopuszczać do zbytznego zmniejszenia się tej liczby (1000 stanowi już wielkość alarmującą).

Lo odzwierciedla tryb małych liter. W tym trybie litery alfabetu zapisują się jako małe, a z klawiszem SHIFT – jako duże. Po użyciu klawisza CAPS, **Lo** zmieni się na **Up** i wówczas klawisze liter dają znaki duże, a z SHIFT – małe. Ten wskaźnik może ponadto być "podświetlony", wówczas wpisywane znaki pojawiają się w negatywie. **Ins** wskazuje na tryb rozsuwania tekstu, **Ovr** w tym miejscu mówi o trybie "zamazywania".

Line=12 i **Col=25** opisują pozycję (wiersz i kolumnę) kursora w redagowanym tekście.

2.2. Redagowanie tekstu

Wybór opcji **Edit** z głównego menu uruchamia mrugający kwadracik (zwany kursorem) w oknie edytora, co umożliwia wprowadzanie lub poprawianie tekstu. Wystukiwane na klawiaturze znaki (z wyjątkiem specjalnych, opisanych niżej) pojawiają się w miejscu, które wskazuje kursor. Trzeba przy tym pamiętać, że długość wiersza nie może przekroczyć 64 znaków. Wprowadzenie do edytora pliku o innym formacie może być przyczyną kłopotów. QA reaguje na zbyt długi wiersz w ten sposób, że wstawia znak End-Of-Line, aby ten wiersz podzielić. W skrajnym przypadku może to spowodować przepełnienie pamięci QA.

Zakończenie wiersza w miejscu wskazanym przez kursor możliwe jest z pomocą klawisza RETURN. W trybie wstawiania (**Ins**) powoduje to podział wiersza na dwie części: część na lewo od kursora zostaje, zaś kursor wraz z resztą tekstu przeskakuje do następnej linii. W trybie zamazywania (**Ovr**) RETURN powoduje tylko przeskok kursora na początek następnego wiersza (o ile jest taki).

Obsługa edytora przypomina w znacznej mierze standard programu SideKick. Ponieważ jest wiele edytorów działających podobnie, które powołują się zwykle na pokrewieństwo z WordStarem, więc i tu nazwano ten sposób pracy "trybem WS". Nie należy przez to rozumieć, że polecenia są takie same, jak w programie WordStar, podobny jest natomiast sposób ich wydawania. Rozkazy dla edytora wydaje się przy pomocy klawisza CONTROL i jakiejś literki, uzupełnionej jeszcze w razie potrzeby dodatkową literą. Działają również zwyczajowe klawisze specjalne ATARI, dublując przeważnie rozkazy trybu WS. Oto ich wykaz:

| rozkaz | ATARI | WS | |
|------------------------------|-----------|------|----|
| kursor w lewo | ^ + | ^ S | |
| kursor w prawo | ^ * | ^ D | |
| kursor do góry | ^ _ | ^ E | |
| kursor w dół | ^ = | ^ X | |
| o stronę do góry | ! ^ _ | ^ R | |
| o stronę w dół | ! ^ = | ^ C | |
| na początek wiersza | | ^ A | *) |
| na koniec wiersza | | ^ Z | *) |
| na początek tekstu | | ^ Qr | |
| na koniec tekstu | | ^ Qc | |
| na początek bloku | | ^ Qb | |
| na koniec bloku | | ^ Qk | |
| do przystanku tabulacji | TAB | ^ I | |
| usunięcie znaku pod kursorem | ^ DELETE | ^ G | |
| usunięcie znaku z lewej | BACKSPACE | ^ H | |
| usunięcie reszty wiersza | | ^ Qy | |
| usunięcie wiersza | !DELETE | ^ Y | |
| wstawienie pustego wiersza | !INSERT | ^ N | |
| zaznaczenie początku bloku | | ^ Kb | |
| zaznaczenie końca bloku | | ^ Kk | |
| powielenie bloku | | ^ Kc | |
| przeniesienie bloku | | ^ Kv | |
| wymazanie bloku | | ^ Ky | |
| wczytanie bloku | | ^ Kr | |
| zapisanie bloku | | ^ Kw | |
| ukrycie/pokazanie bloku | | ^ Kh | |

| rozkaz | ATARI | WS |
|--------------------------------|---------------------|-----------------|
| poszukiwanie nowego tekstu | | [^] Qf |
| kontynuowanie poszukiwania | | [^] L |
| przełączenie Ins/Ovr | [^] INSERT | [^] V |
| przełączenie Lo/Up | CAPS | |
| przełączenie poz/neg | LOGO | |
| ustawienie/kasowanie tabulacji | [^] TAB | |
| wyświetlenie znaku specjalnego | | [^] P |
| opuszczenie edytora | ESC | [^] Kd |

Uwaga: ze względu na brak niezbędnych w pracy klawiszy **Home** i **End** (dostępnych na komputerach PC) użyto niezgodnych z WS (ale wygodnych) kombinacji oznaczonych *).

Rozkaz poszukiwania tekstu [^]Qf wymaga wprowadzenia wzoru do poszukiwania. Pojawia się specjalna ramka, gdzie wpisuje się ów wzór, kończąc klawiszem RETURN. Jeśli przy tym naciśnięty był klawisz SHIFT, to poszukiwanie przebiega od początku tekstu, w przeciwnym razie – od miejsca, w którym stoi kursor. Duże i małe litery nie są rozróżniane. Rozkaz [^]L pozwala ominąć etap wprowadzania wzoru: poszukiwany jest uprzednio podany tekst, zawsze od pozycji kursora. Uwaga: w rzeczywistości poszukiwanie rozpoczyna się od następnego znaku po kursorze. Wobec tego wystąpienie wzoru na pozycji kursora jest ignorowane.

Blokiem nazywać będziemy fragment tekstu oznaczony poprzez użycie rozkazów [^]Kb i [^]Kk odpowiednio na jego początku i końcu. Blok jest widoczny (aktywny, wówczas mruga), lub ukryty, co można przełączać za pomocą rozkazu [^]Kh. Początek bloku zaznacza się stojąc kursorem na jego pierwszym znaku, zaś koniec – tuż po ostatnim znaku (jeśli ostatnim znakiem bloku ma być znak końca wiersza, trzeba stanąć w pierwszej kolumnie następnej linijki). Znacznik bloku ustawiony gdziekolwiek po ostatnim znaku w wierszu zostanie przesunięty na jego koniec, tuż przed EOL, w związku z usuwaniem przez edytor QA spacji kończących wiersz.

Dla wykonania operacji na bloku potrzeba, aby był on aktywny.

Rozkazy **^Kc** i **^Kv** kopiują blok w pozycji wskazanej przez kursor. **^Kv** dodatkowo wymazuje ten blok w miejscu jego dotychczasowego pobytu. Rozkaz **^Ky** usuwa blok.

Rozkazy **^Kr** i **^Kw** wymagają podania nazwy pliku. Zasady są takie same, jak przy omawianych wyżej **Load** i **Save**. **^Kr** umieszcza wskazany plik jako blok w miejscu wskazanym przez kursor. **^Kw** pozwala zapisać oznaczony blok w pliku o wybranej nazwie.

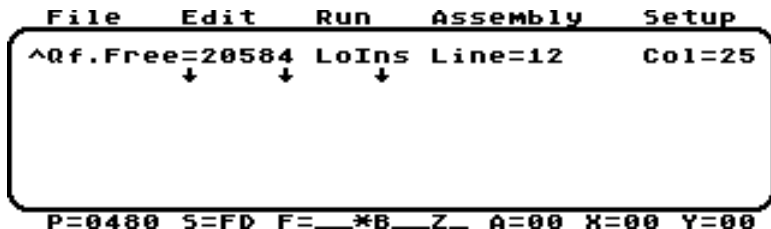
Należy mieć tu na uwadze, że do wykonania operacji blokowych edytor potrzebuje więcej pamięci, niż to wynika z rozmiaru bloku. Jeśli pamięci jest za mało, to operacja się nie wykona. Trzeba też pamiętać (a najpierw wiedzieć), że wiersz, w którym stoi kursor, przetwarzany jest poza buforem tekstu, a uaktualniany dopiero w chwili, gdy kursor opuszcza tę linię lub wykonywana jest jakaś specjalna operacja. W związku z tym również informacja o wolnej pamięci zmienia się skokowo we wspomnianych momentach, a nie przy wpisywaniu lub usuwaniu pojedynczego znaku. Klawisz RESET lub wyczerpanie wolnej pamięci powoduje w związku z tym na ogół przywrócenie poprzedniego stanu bieżącego wiersza.

3. Run

Run powoduje wykonanie programu maszynowego od adresu, który widnieje w ostatnim wierszu ekranu pod hasłem **P=**. Ten wiersz opisuje stan procesora, z jakim podejmie on wykonywanie programu.

Nad wykonaniem programu czuwa mini-debugger, który przechwytuje przerwania od rozkazu BRK i od klawisza BREAK, podmienia rejestry procesora, ekran, stronę zerową, stos i parę istotnych dla QA komórek, aby nie dać się testowanemu programowi wysadzić z siodła, a zarazem stworzyć mu takie warunki, jakby był wywołany bezpośrednio przez DOS.

P= to wskaźnik programu, **S=** to wskaźnik stosu, **F=** to rejestr znaczników, **A=**, **X=** i **Y=** to rejestry danych. Wszystkie liczby podane są w notacji szesnastkowej. Znaczniki przedstawiono za pomocą stosownych liter (lub ich braku).



Uruchomiony program zobowiązany jest powrócić do QA, a może to zrobić dobrowolnie na jeden z trzech sposobów: rozkazem **RTS**, rozkazem **JMP (10)**, lub rozkazem **BRK**. Jeśli program nie przejawia ochoty do powrotu, można to wymusić kombinacją klawiszy **!BREAK**. W przypadku, gdyby i ta metoda zawiodła (program może dezaktywować klawisz **BREAK**), ostatnią deską ratunku pozostaje **RESET**. Trzeba jednak pamiętać, że wiele DOS-ów zachowuje się podczas "RESETowania" co najmniej dziwnie i trudno przewidzieć co zostanie z tekstu po takiej operacji. Oczywiście system zawarty na oryginalnej dyskietce z QA gwarantuje poprawne działanie programu w każdej sytuacji.

Rozkaz **BRK** "zamraża" stan procesora tak, że przy powtórnym wykonaniu **Run** procesor podejmie pracę od miejsca przerwania. Podobnie dzieje się w przypadku zastosowania **!BREAK**. **RTS** natomiast, **JMP (10)** oraz **RESET** przywracają stan początkowy procesora: rejestry są zerowane, stos zawiera tylko adres powrotu do QA, wskaźnik programu przyjmuje pewną z góry ustaloną wartość (patrz: **Setup**). W chwili wywołania testowanego programu QA oddaje do dyspozycji użytkownika jego własny ekran, kolory, generator znaków, stronę zerową i stos. Po powrocie QA przechowuje te dane, tak, że pozostaną nie

zmienione aż do następnego **Run**. Wynika z tego, że testowany program może zmieniać je bez obawy o "zdrowie" QA. Oczywiście na stronie zerowej obowiązuje pewien porządek wymuszony przez system. Dla użytkownika pozostawia się adresy od \$80 do \$FF (od \$CB z BASIC-em). W przypadku, gdy testowany program instaluje własną procedurę obsługi przerwania i wraca do QA, to procedura owa może użyć tylko obszaru \$F0..\$FF i nie powinna manipulować wymienionymi wyżej, przechowywanymi danymi w czasie, gdy pracuje QA. Komórki \$F0..\$FF nie są przez QA nigdy używane.

Podczas pracy z QA można zawsze zajrzeć na "ekran użytkownika" za pomocą kombinacji klawiszy **!^Spacja**. Dowolny klawisz przywraca widok QA. Jest to pomocne przy porównywaniu efektów działania programu z jego tekstem.

Uwaga: wektory przerwania klawisza BREAK i rozkazu BRK są wymieniane na czas wykonywania programu, zaś z chwilą powrotu do QA przywraca się im stare wartości. Wynika stąd, że nie warto (i nie należy) używać ich do innych celów.

Z dużą rozważą należy używać rezydentnych monitorów pamięci takich, jak wglądownica w programie XLFriend. Problem polega na wymianie przez QA strony zerowej, stosu i niektórych innych komórek na czas wykonania programu. XLF wywołany podczas działania testowanego programu pokaże zatem całkiem inne wartości niż po powrocie do QA. Pochopna próba zmiany jakichś bajtów, zwłaszcza na stronie zerowej, gdy program już nie działa, może zakończyć się dużym kłopotem.

4. Assembly

Ta opcja służy do asemblowania tekstu zawartego w pamięci edytora. W dolnym wierszu ekranu zamiast stanu procesora pojawia się informacja o przebiegu asemblacji. **Pass=** informuje o przebiegu (są dwa), a **Line=** pokazuje numer aktualnie przetwarzanego wiersza programu. Prócz tego w opróżnionym chwilowo oknie edytora może pojawić się "listing" w zgodzie z opcjami asemblacji ustalonymi w tekście programu. Jeśli wyświetlana informacja nie mieści się w oknie edytora, asembler przerywa pracę, czekając na naciśnięcie dowolnego klawisza. Asemblacja może też spowodować odczyt plików wskazanych rozkazami **ICL**, zapis różnych plików na różne urządzenia zewnętrzne i (lub) kodu do pamięci. Asembler

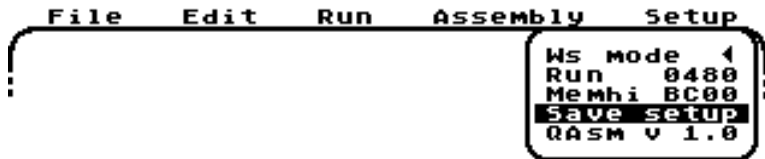
kończy pracę sygnałem dźwiękowym (jeśli wystąpiły błędy) lub po cichu, co oznacza sukces. Końcowy stan ekranu assemblera utrzymany jest aż do pierwszego użycia klawiatury.

Asemlacja wymusza przestawienie rejestrów procesora w stan początkowy.

Umieszczenie kodu w pamięci możliwe jest w dwóch obszarach. Krótkie programy mogą siedzieć między adresem \$480 a \$6FF. Dlatego domyślny adres, pod który QA wykonuje skok przy użyciu opcji **Run** jest właśnie \$480. W przypadku większego programu trzeba obniżyć górną granicę pamięci używanej przez QA, tak, jak to jest opisane w ustępie 5.3, a kod programu usadowić pomiędzy ustalonym adresem, a ekranem.

5. Setup

Wybranie tej opcji powoduje otwarcie dodatkowego menu:



5.1. Ws mode

Trójkątny "ptaszek" stojący przy Ws mode określa, że w edytorze działają rozkazy w stylu WordStara. Wybranie tej opcji powoduje przełączenie edytora w tryb, w którym działają tylko klawisze ATARI (patrz: **Edit**). Wtedy "ptaszek" znika. Powtórne wybranie tej opcji przywraca tryb **WS**.

5.2. Run

Run 0480 określa adres, który wstawiany jest do wskaźnika programu procesora, gdy ustawiane są wartości początkowe rejestrów (patrz: **Run**). Wybranie tej opcji pozwala na wpisanie nowego adresu (szesnastkowo). Krótkie programy można umieszczać bezpośrednio od \$480. Dłuższe można uruchamiać zmieniając odpowiednio ten właśnie adres lub wpisując pod adresem \$480 rozkaz skoku do początku właściwego programu. Wpisywanie adresu kończy się klawiszem RETURN bądź przerywa przez ESC.

5.3. Memhi

Memhi BC00 wskazuje na pierwszy wolny bajt powyżej obszaru pamięci zajmowanego przez QA. Domyślnie jest to tuż pod pamięcią ekranu użytkownika. Adres ten można zmienić (tylko starszy bajt), podobnie jak adres **Run**, zalecana jest wszakże duża ostrożność, bo zwiększenie go spowoduje zniszczenie pamięci ekranu, zmniejszenie zubaża pamięć edytora (**Free=**). Manipulowanie tym adresem służy do wygospodarowania wolnej pamięci potrzebnej, być może, dla testowanego programu. Adres zapisuje się w notacji szesnastkowej. W przypadku wpisania liczby spoza dopuszczalnego zakresu QA samoczynnie ją skoryguje. Możliwe jest w krytycznych przypadkach zagospodarowanie także pamięci ekranu systemowego poprzez podniesienie **Memhi** na granicę ROM-u. QA bowiem z tego obszaru zwykle nie korzysta.

5.4. Save setup

D Save setup pozwala zapisać poczynione ustalenia w pliku QA.SET, na dyskietce, w napędzie, z którego uruchomiono QA (nie ma możliwości zmiany tej nazwy ani urządzenia). Podczas uruchamiania QA poszukuje tego pliku i jeśli znajdzie, to ustawia się zgodnie z zapisanymi wytycznymi. Oprócz adresów **Run** i **Memhi** zapamiętane zostaje ustawienie edytora: tryb WS/ATARI, Lo/Up (klawisz CAPS), pozytyw/negatyw (klawisz LOGO), **Ins/Ovr** oraz mapa tabulacji. Dyskietka dystrybucyjna zawiera przykładowy plik QA.SET, który ustawia adresy Run i Memhi na \$8800, jak wymagają tego programy przykładowe (opisane w ostatniej części podręcznika).

C Nie używa się. Jednakże niektóre systemy "Turbo", jak np. Blizzard, definiują urządzenie "D:". W takim przypadku QA po uruchomieniu spróbuje odczytać plik D:QA.SET (jeszcze przed ukazaniem się menu). Trzeba wówczas spowodować błąd odczytu, np. naciskając BREAK (można oczywiście) zapisywać i odczytywać ten plik tak, jak na dyskietce, ale więcej z tym kłopotu niż pożytku).

5.5. Qasm

Ostatnia linijka tego menu jest tylko do czytania: podaje ona numer wersji QA.

BUG HUNTER

Bug Hunter (w skrócie BH) jest nowoczesnym debuggerem "całoe ekranowym", co oznacza, że współpraca z nim nie polega na wymianie pytań i odpowiedzi, lecz można śledzić w sposób ciągły zachowanie się testowanego programu. Ekran pokazuje nieustannie wszystkie ważne punkty komputera: rozkazy wykonywanego programu, rejestry procesora, fragment (dowolny) pamięci operacyjnej oraz stos. Możliwość wykonywania rozkazów w różnym tempie, mechanizm "pułapek", czytelna prezentacja informacji oraz wygodna obsługa czynią z BH niezastąpione narzędzie do podglądania pracy komputera.

D Aby uruchomić BH z oryginalnej dyskietki, należy po wyświetlonej przez DOS zachęcie "D1:" napisać
BH
i nacisnąć RETURN.

C Aby uruchomić BH z oryginalnej kasyety, należy ustawić ją w miejscu, gdzie znajduje się ten program, po wyświetlonej przez COS zachęcie ">" napisać
**
i nacisnąć RETURN

Uwaga: systemowy wskaźnik MEMLO (zawarty w słowie \$2E7, a możliwy do odczytania w DOS-ie lub COS-ie rozkazem MEM) nie może przekraczać \$4800!

BH zgłasza się numerem wersji i komunikatem "Press SHIFT/BREAK to get control", co oznacza, że pozostanie on w uśpieniu aż do chwili użycia !BREAK. Wówczas aktualnie działający program (np. gra, DOS, czy interpreter BASIC-u) zostaje przerwany, pojawia się natomiast ekran BH ze szczegółowym

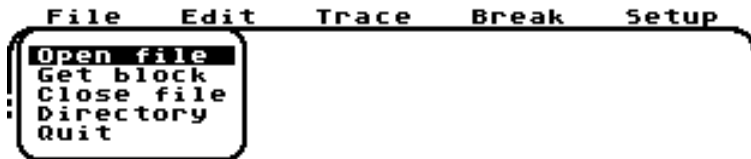
odzwierciedleniem stanu procesora w chwili przerwania. Obraz widoczny poprzednio nie przepada. Można go oglądać przy pomocy klawisza spacji. Górny wiersz ekranu zawiera główne "menu" BH.



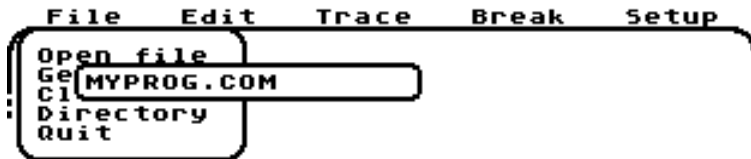
Jedna z opcji wyróżniona jest podświetleniem, można je przemieszczać w prawo lub w lewo, naciśnięcie zaś RETURN powoduje wywołanie podświetlonej funkcji. Innym, nie gorszym, sposobem aktywizacji opcji jest użycie pierwszej litery jej nazwy. Powrót do głównego menu umożliwi klawisz ESC.

1. File

Otwiera się dodatkowe okno z opcjami:



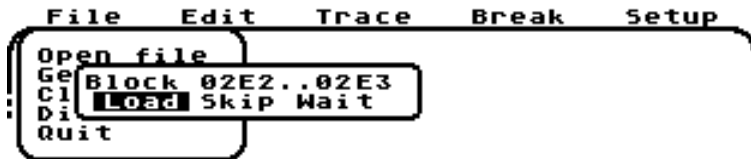
Można je aktywizować przez wskazanie ruchomym podświetleniem i naciśnięcie RETURN, bądź bezpośrednio za pomocą pierwszej litery. Open file pozwala wskazać plik zawierający program binarny w formacie DOS-u lub w formacie kasetowym ("boot"), który będzie analizowany. BH nawiązuje kontakt ze wskazanym plikiem poprzez standardowe procedury WE/WY, przy użyciu tzw. bloku IOCB (trzeba w związku z tym pamiętać o zamknięciu pliku przed rozpoczęciem śledzenia programu korzystającego z WE/WY). Nazwę pliku podaje się w dodatkowym okienku:



D Jeżeli nazwa pliku zawiera znaki "*" lub "?", BH przedstawia spis plików z dyskiety, odpowiadających podanemu szablónowi. Wówczas, używając strzałek w górę i w dół, można wybrać żądany plik i zatwierdzić wybór klawiszem RETURN.

C Plik ten musi zawierać długie przerwy między rekordami, ponieważ BH wczytuje program po kawałku.

Get block jest poleceniem wczytania jednego segmentu programu z pliku, który został "otwarty" wyżej opisanym sposobem. Jeżeli żaden plik nie był wcześniej otwarty, BH wykonuje tu wszystkie czynności opisane pod **Open file**. Segment (blok) jest logiczną częścią pliku w formacie DOS-u, ciągiem bajtów, które DOS umieszcza pod adresem określonym w początkowych bajtach tego bloku, zwanych nagłówkiem. BH czyta nagłówek i przedstawia propozycje do wyboru, np.:



Można wybrać spośród: **Load** – umieszczenie bloku w pamięci, **Skip** – ominięcie tego bloku, **Wait** – chwilowe zaprzestanie operacji **Get block**. Wyboru dokonuje się ruchomym podświetleniem lub za pomocą pierwszej litery. Uwaga: jeśli blok miałby się ulokować w obszarze zajęтым przez BH, opcja **Load** się nie pojawi. Wczytany blok może ustawiać adres inicjalizacji, co powodowałoby w DOS-ie wykonanie fragmentu programu. W BH trzeba to zrobić świadomie (patrz: **Trace**), można też zrezygnować. Plik pozostaje otwarty i gotowy do wczytania następnego bloku, aż do chwili zamknięcia go poprzez **Close file**, fizycznego końca danych (komunikat "End of file") lub otwarcia nowego pliku opcją **Open file**.

D | Directory umożliwia obejrzenie spisu plików zawartych na dyskietce.

Quit kończy pracę BH. Zwraca on wektory przerwań, zwalnia pamięć i powraca do DOS (COS).

2. Edit

Wybór tej opcji powoduje uaktywnienie szczególnego edytora udostępniającego do manipulacji serce komputera - procesor

| File | | | Edit | | Trace | | Break | | Setup | |
|------|-----|------|------|--|---------------|--|-------|--|-------|--|
| 14C7 | LDA | #05 | | | registers | | 35 | | | |
| 14C9 | STA | 2600 | | | SP=01FD | | 24 | | | |
| 14CC | LDX | D4 | | | PC=14C7 | | FF | | ▶ | |
| 14CE | BNE | 14D3 | | | F=N_*B____C | | E4 | | | |
| 14D0 | JSR | F556 | | | A=01 | | 00 | | | |
| 14D3 | JMP | E474 | | | X=C5 | | 00 | | | |
| 14D6 | BRK | | | | Y=3F | | 00 | | | |
| 14D7 | BRK | | | | | | 00 | | | |
| 14D8 | BRK | | | | Memory 1C3A | | 00 | | | |
| 14D9 | BRK | | | | 61626364 ABCD | | 00 | | | |
| 14DA | BRK | | | | 33323130 3210 | | 00 | | | |
| 14DB | BRK | | | | 20212022 ! " | | 00 | | | |
| 14DC | BRK | | | | 23202420 # \$ | | 00 | | | |

Kursor pojawia się w części dotyczącej rejestrów procesora, umożliwiając ich modyfikację. Można przesuwac kursor przy pomocy klawiszy -, =, + i *, a wartości poprawiac poprzez wpisanie cyfr szesnastkowych dla rejestrów liczbowych, lub klawiszem LOGO dla rejestru znaczników. Poniżej rejestrów wyświetlony jest fragment pamięci od adresu zatytułowanego memory. Adres ten można zmieniać podobnie, jak zawartość rejestrów. Pamięć ukazywana jest w postaci liczb szesnastkowych i ich odpowiedników znakowych. Znaki mogą odpowiadać kodom ASCII lub "ekranowym" w zależności od ustawienia (patrz: Setup). Wzdłuż prawej krawędzi ekranu pokazano fragment pierwszej strony pamięci, obok którego trójkątny "ptaszek" zaznacza aktualne położenie wskaźnika stosu. Stos zwrócony jest wierzchołkiem w dół, zaś adres \$1FF (dno stosu) jest u góry ekranu. Jeśli stos zapelni się bardziej, niż można to pokazać w 20 wierszach, część jego zawartości schowa się poza ekranem, tak, aby "ptaszek" zawsze był widoczny. Lewą część ekranu zajmują rozkazy śledzonego aktualnie programu przedstawione w formacie języka asemblera z tym, że liczby przedstawione są w notacji szesnastkowej, bez używanego zwyczajowo

znacznika "\$". Poziome rozjaśnienie, ściśle związane z rejestrem PC procesora, wskazuje rozkaz przeznaczony do wykonania. Rozjaśnienie to można przesuwac klawiszami ^- i ^= o jedną pozycję lub klawiszami < i > o całą długość ekranu, powodując zarazem odpowiednie zmiany PC. I odwrotnie: każda zmiana PC spowoduje przemieszczenie rozjaśnienia na żądany rozkaz.

3. Trace

Trace powoduje wykonanie śledzonego programu, zaczynając od takiego stanu procesora, jaki widnieje na ekranie. Na czas wykonywania rozkazów testowanego programu BH oddaje mu stronę zerową, stos, pamięć ekranu, program ANTIC-u, kolory, które przechował podczas konwersacji z operatorem. Zatrzymanie wykonywania programu może nastąpić poprzez:

- naciśnięcie klawiszy !BREAK
- napotkanie rozkazu BRK
- wystąpienie warunku wyjątkowego (patrz: **Break**)

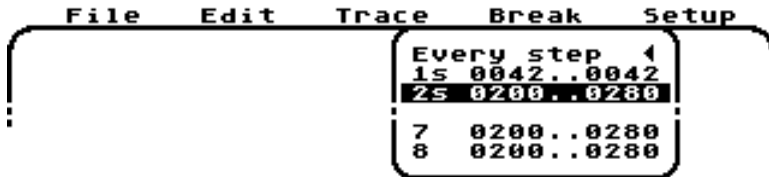
Podczas wykonywania testowanego programu można przełączać ekran na używany przez program klawiszami ^Spacja lub na używany przez BH klawiszami !Spacja. Uwaga: wybranie opcji **Trace** przy naciśniętym klawiszu SHIFT spowoduje bezpośredni wskok do programu, bez śledzenia i bez kontroli warunków (program wykonuje się bez udziału BH). Przerwanie programu jest wówczas możliwe tylko przez !BREAK lub BRK, a czasem to i wcale.

BH umożliwia śledzenie programu także w ROM-ie komputera. Oglądanie procedur napisanych przez naszych amerykańskich kolegów po fachu daje nieraz wiele do myślenia. Trzeba jednak pamiętać, że "puszczenie" systemu od miejsca wybranego na oko prowadzi przeważnie do zablokowania komputera. W szczególności BH potrzebuje nieraz kontaktu z klawiaturą, a jeśli śledzony program wyłączy, powiedzmy, przerwania... Taka zabawa wymaga od użytkownika wiedzy i uwagi. W pewnych typowych sytuacjach BH sam ostrzega przed niebezpieczną operacją, jak każde jednak mocne narzędzie bywa także niebezpieczny.

W przypadku podglądania programu wprowadzonego do pamięci RAM należy na wstępie określić jego adres uruchomienia. Uzyskuje się to w przypadku formatu DOS-owego poprzez ustawienie w Edit adresu oglądanej pamięci (memory) na 2E0, gdzie kolejne dwa adresy pokazują główne wejście do programu i procedurę inicjalizacji. Odpowiedni adres trzeba przepisać do PC. Dla formatu kasetowego, który poznać po tym, że podczas czytania zamiast słowa Block widnieje Boot, informacji o adresach należy szukać w pierwszych sześciu bajtach wczytanego pliku. Szczegóły dotyczące budowy plików binarnych są zresztą powszechnie znane.

4. Break

Ta opcja służy do zastawiania "pułapek", czyli miejsc lub warunków, w których wykonywanie programu ma zostać przerwane. Dodatkowe menu pozwala ustawić te warunki.



Trójkątny "ptaszek" przy Every step wskazuje, że program ma być przerywany po wykonaniu każdego rozkazu. Wybranie tej opcji włącza/wyłącza tryb zwany powszechnie pracą krokową. Pozostałe opcje opatrzone numerami od 1 do 8 umożliwiają zastawienie pułapki w pewnym obszarze (zakresie) pamięci. Literka stojąca tuż obok numeru oznacza typ pułapki, czyli sytuację, w której wykonywanie programu zostanie przerwane:

- a (access) – próba dostępu do komórki o podanym zakresie adresów
- s (store) – próba zapisu do komórki o podanym zakresie adresów
- t (trace) – próba wykonania rozkazu leżącego w podanym zakresie

Brak literki oznacza pułapkę "wyłączoną", to znaczy, że BH nie sprawdza zakresu podanego w tym wierszu. Warto zanotować, że od liczby aktywnych pułapek zależy maksymalna prędkość BH.

Aby ustawić pułapkę, należy wskazać ją ruchomym podświetleniem i nacisnąć RETURN (lub użyć bezpośrednio klawisza ze stosowną cyfrą). W pierwszej kolejności ustawia się typ pułapki, używając dla jego zmiany klawisza LOGO, a dla zaakceptowania wyboru – RETURN. Następnie redaguje się kolejno dwie liczby szesnastkowe: początek i koniec zakresu. Użycie klawisza ESC lub – lub = pozwala przerwać ustalanie w dowolnej chwili, ale wprowadzone dotąd zmiany pozostają. Dla podpatrzenia, na przykład, odwołań do dysku należy ustawić pułapkę typu t na E459..E459 lub typu s na 0300..030F. Niektóre sytuacje powodują przerwanie wykonania programu w podobny sposób, lecz niezależnie od opisanych tu pułapek, Są to:

- rozkaz BRK
- próba modyfikacji pamięci używanej przez BH
- próba wykonania nieistniejącego (nielegalnego) rozkazu
- przepełnienie stosu

Rozkaz BRK oraz warunki pracy krokowej są traktowane jako normalne (przewidziane) przerwania. Pozostałe warunki oraz pułapki wyświetlają komunikaty o przyczynie przerwania, np.:



```
User  store  trap
Abort Skip Run
```

Komunikat pojawia się w prawym dolnym kącie ekranu aby zasłaniać jak najmniej miejsc ciekawych dla obserwatora i pozwala niekiedy wybrać spośród możliwości: **Abort** - powrót do menu, **Skip** - ominięcie krytycznego rozkazu, **Run** - kontynuowanie programu. Czasem jednak kontynuowanie jest niemożliwe, wtedy BH prosi tylko o przeczytanie komunikatu, a do wyboru jest jedna możliwość: **Ok**, co powoduje bezwarunkowy powrót do menu.

5. Setup

Ta opcja przywołuje dodatkowe menu, w którym zlokalizowane są różne przełączniki ("ptaszek" stojący w danej linii oznacza opcję włączoną, zaś jego brak – wyłączoną):



Delay służy do ustalenia zwłoki pomiędzy wykonaniem kolejnych rozkazów śledzonego programu. Wybranie tej opcji pozwala wprowadzić opóźnienie w postaci liczby szesnastkowej od 00 do FF. Zakończenie klawiszem RETURN lub ESC. 00 oznacza brak opóźnienia i wtedy szybkość śledzonego programu zależy tylko od żwawości BH. Warto pamiętać, że praca z widocznym ekranem BH jest znacznie wolniejsza (z racji wyświetlania obrazu) niż wtedy, gdy widać tylko ekran "użytkownika". To zjawisko nie występuje przy niezerowej wartości opóźnienia.

Synch powoduje, że adres pamięci pokazywanej pod hasłem memory w polu edytora odpowiada efektywnemu adresowi pod jakim bieżący rozkaz odwołuje się do pamięci, w przeciwnym razie adres ustawiany jest ręcznie.

Panel, to inaczej ekran BH. Gdy się go wyłączy, to nie widać ekranu BH podczas działania testowanego programu. Tę jedynie opcję można zmieniać "zdalnie", podczas trwania podpatrywanego programu, naciskając klawisze !Spacja lub ^Spacja. Trzeba wszakże rozumieć, że dla poprawnego działania testowanego programu BH zawsze wyłącza panel na czas wykonywania każdego rozkazu. Aby nie powodować migotania obrazu, przełączenia te, a zatem i rozkazy, są zsynchronizowane z powrotem plamki (wygaszaniem pionowym) w telewizorze. To jednak sprawia, że pętle oczekiwania na pewne warunki, np. na konkretną wartość licznika linii w procesorze ANTIC, mogą się nigdy nie skończyć. Przypadek taki ma miejsce między innymi w systemowej procedurze CLICK, generującej dźwięk związany z naciśnięciem klawisza. W takich sytuacjach konieczne jest wyłączenie panelu (wtedy rozkazy nie będą zsynchronizowane).

Ascii wskazuje, że znaczkii opisujące fragment pamięci w obszarze edytora memory odpowiadają kodom ASCII, w przeciwnym razie są to kody "ekranowe".

Turbo powoduje, że podprogramy zawarte w ROM-ie i wywoływane przez standardowe wektory \$E450..\$E48C wykonują się bez śledzenia, za to z pełną szybkością procesora. Z możliwości tej korzysta się głównie przy testowaniu operacji WE/WY, które nie mogłyby się wykonać w zwolnionym (normalnym dla BH) trybie.

Napis BH 1.0 służy tylko do czytania, podaje numer wersji BH.

XL FRIEND

XL Friend (zwany w skrócie XLF) stanowi zestaw czterech pożytecznych programów, które po zainstalowaniu są stale obecne w pamięci komputera. Są to:

- prosty edytor tekstów
- tabela kodów ASCII, ekranowych i klawiatury
- kalkulator
- monitor pamięci

D Aby uruchomić XLF z oryginalnej dyskietki, należy po wyświetlonej przez DOS zachęcie "D1:" napisać XLF i nacisnąć RETURN.

C Aby uruchomić XLF z oryginalnej kasety, należy ustawić ją w miejscu, gdzie znajduje się ten program, po wyświetlonej przez COS zachęcie ">" napisać ** i nacisnąć RETURN

Uwaga: systemowy wskaźnik MEMLO (zawarty w słowie \$2E7, a możliwy do odczytania w DOS-ie lub COS-ie rozkazem MEM) nie może przekraczać \$3F00!

XLF "chowa się" w pamięci RAM znajdującej się "pod" ROM-em systemu operacyjnego, a w głównej pamięci jest tylko fragment obsługujący przerwanie od klawiatury i zarządzający nakładkowaniem programów zestawu, zajmujący niecałą stronę. XLF nie będzie działał na komputerach nie wyposażonych w dodatkową pamięć (nie XL/XE), nie współpracuje też z programami, które same wykorzystują pamięć

pod ROM-em (np. SpartaDOS, Turbo BASIC XL). Inną przyczyną konfliktów może być zbójcejski sposób korzystania z pamięci przez wiele prymitywnych programów (z DOS-em 2.5 na czele), które nie szanują wskaźników dostępnej pamięci, zwłaszcza MEMLO. Standardem, któremu są wierne nasze wszystkie programy, jest lokowanie się tylko w granicach dostępnej pamięci (programy się relokują!). XLF umieszcza się na granicy MEMLO, a następnie zmienia ten wskaźnik, by ochronić się przed zniszczeniem. Tymczasem DUP DOS-a 2.5 ładuje się na oślep w stałe miejsce pamięci, nie zważając na MEMLO. Jeśli wszakże nie instaluje się RAM-DYSKU, to między DOS-em, a DUP-em pozostaje niewielka szczelina, w sam raz dobra dla XLF-a. Problem ten nie występuje, rzecz jasna, w zestawie DOS 2.5 i Chaos CP. CP jest nakładką eliminującą DUP, sterowaną komendami, napisaną przez programistów związanych blisko z firmą AVA-LON.

Gdy XLF jest nieaktywny, nic nie wskazuje na jego istnienie. Można naciskać RESET, uruchamiać programy, pracować w BASIC-u. Zapominalskim klawisz HELP przypomina o sposobie wywołania programów XLF-a. Jeśli zaś któryś z nich jest aktywny, HELP przypomina o podstawowych funkcjach tego programu. XLF jest jednym z nielicznych programów, które "reinkarnują" po zimnym starcie komputera, o ile nie był on wyłączany i jeśli podczas zimnego startu XLF nie był aktywny. Podczas instalowania XLF przeszukuje pamięć i, jeśli może, odtwarza swój stan z poprzedniej sesji. Jeżeli z jakichś powodów nie jest to pożądane, można wymusić załadowanie "świeżego" XLF-a, trzymając naciśnięty klawisz SELECT, gdy na ekranie widnieje napis "Checking...". Powtórne wczytanie XLF-a pozwala usunąć go z pamięci, o ile nie został "przygnieciony" innymi nakładkami.

1. Edytor

Edytor wywołuje się kombinacją klawiszy !^1. Powoduje to ukazanie się prostokątnej ramki. Na obrzeżu widnieją informacje o stanie edytora. Napis **Edit** wskazuje na tryb redagowania. Zainicjowanie operacji WE/WY powoduje zmianę napisu na **Load** lub **Save**. Wówczas można poprawić i zaakceptować nazwę pliku widoczną tuż obok. W skład nazwy pliku musi wchodzić specyfikacja urządzenia, np. C:, P:, itd. W przypadku wystąpienia błędu WE/WY pojawia się napis **Fail** i numer błędu. Liczba w prawym górnym rogu pokazuje kod ASCII znaku, na którym stoi kursor. Dolna krawędź ramki zawiera informacje o zasobie

wolnej pamięci i współrzędnych kursora. Redagowanie tekstu polega na wprowadzaniu znaków z klawiatury. Poruszanie się kursorem umożliwiają cztery klawisze opatrzone strzałkami. Prócz tego istnieją rozkazy specjalne, wprowadzane przez jednoczesne naciśnięcie dwóch lub trzech klawiszy:

| | |
|---------|---------------------------------------------------------|
| ^A | skok na początek bieżącego wiersza |
| ^Z | skok na koniec bieżącego wiersza |
| ^B | skok na początek tekstu (Begin) |
| ^E | skok na koniec tekstu (End) |
| ^Q | rozkaz uznania następnego klawisza jako znaku (Quote) |
| ^U | zmniejszenie okna edytora (Up) |
| ^D | powiększenie okna edytora (Down) |
| ^L | wczytanie tekstu z urządzenia WE/WY (Load) |
| ^S | zapisanie tekstu na urządzeniu WE/WY (Save) |
| ^T | ustawienie znacznika w bieżącym wierszu (Tag) |
| ^G | przeskok do zaznaczonego wiersza (Go to tag) |
| ^V | przełącznik widzialności znaków EOL (Visibility) |
| ^I | przeniesienie do edytora fragmentu ekranu (Import) |
| !INSERT | wstawienie wskazanego kursorem wiersza przed zaznaczony |
| ^DELETE | usunięcie znaku na pozycji kursora |
| !DELETE | usunięcie wiersza wskazanego kursorem |
| !^E | wymazanie całego tekstu (bezpowrotne!) (Erase) |
| !^= | skok o długość okna do przodu |
| !^~ | skok o długość okna wstecz |
| ESC | powrót do przerwanego programu |
| HELP | przypomnienie podstawowych funkcji |

Długość wiersza ograniczona jest do 128 (127 znaków tekstu + znak EOL). Pewnego omówienia wymaga przenoszenie tekstów. Wewnątrz edytora kopiować można po jednym wierszu z miejsca, gdzie stoi kursor do wiersza oznaczonego dużą kropką na pionowym brzegu ramki. Kropkę ustawia się za pomocą ^T.

Natomiast z ekranu przykrytego ramką edytora (z uwzględnieniem marginesów trybu tekstowego OS) przenosi się tekst w miejsce wskazane kursorem przez naciśnięcie ^I. Szybkość importowania zależy od pozycji kursora w tekście, ponieważ przenoszone znaki przepychają tekst przed sobą.

Uwaga: edytor nie ostrzega o możliwości utraty nie zachowanego tekstu.

2. Tabela kodów

Narzędzie to wywołuje się klawiszami !^2. Prostokątna ramka tabeli zawiera zestaw znaków ATARI. Znak wskazany kursorem opisany jest na dolnej kresce ramki: pod hasłem "A" jako kod ASCII i "I" - kod ekranowy (internal). Na górnej kresce natomiast widnieje kod matrycowy ostatnio naciśniętego klawisza. TAB pozwala przełączać tryb wyświetlania kodów z dziesiętnego na szesnastkowy i z powrotem. Wybór znaku za pomocą klawiszy +, *, -, =. Te same klawisze wspólnie z CONTROL umożliwiają przesunięcie tabeli, gdyby zasłaniała jakiś istotny fragment ekranu. Klawisz HELP przypomina najważniejsze informacje.

3. Kalkulator

Liczydłko to, wywoływane naciśnięciem !^3, operuje na liczbach całkowitych z przedziału 0..65535 (najważniejszych z punktu widzenia programisty). Obrazek kalkulatora zawiera zarazem opis jego klawiszy funkcyjnych. Oprócz czterech działań (dzielenie niestety z obcięciem ułamka) pozwala także na operacje logiczne na bitach: OR (^O), AND (^A), EOR (^E). Możliwe są operacje na pamięci: !+ (dodawanie) i !- (odejmowanie) oraz ^M (przywołanie jej zawartości). Można też zmieniać tryb wyświetlania liczb na: dziesiętny (^D), szesnastkowy (^H) lub dwójkowy (^B). Informacja o ustawionym trybie widnieje w górnej części ramki. Gdyby kalkulator zasłaniał coś ważnego na ekranie, można go przesunąć klawiszami ^+, ^*, ^-, ^=. HELP przypomina najważniejsze informacje.

4. Monitor pamięci

Monitor aktywizuje się poprzez !^4. Program ten, zwany powszechnie "wglądownicą", pozwala na obserwowanie dowolnego fragmentu pamięci komputera w czasie rzeczywistym, co sprawia, że widać na

bieżąc wszelkie zmiany, np. zegar w komórkach 18, 19, 20. W lewej części dane są pokazywane w formie liczb szesnastkowych, jedna z nich wyróżniona jest ruchomym kursorem, który można przesuwać klawiszami +, *, -, = o jedną pozycję w odpowiednim kierunku, lub klawiszami <, > o 256 bajtów (stronę) odpowiednio w dół lub w górę pamięci. W prawej części okna widnieją znaki odpowiadające pokazywanym bajtom pamięci. Znaki są przedstawiane w kodzie ASCII lub "internal", co uwidocznione jest w górze ramki, a przełącza się klawiszem TAB. Dolna kreska ramki zawiera adres wskazany kursorem: w notacji dziesiętnej i szesnastkowej. Bajt podświetlony kursorem można zmienić, wpisując (szesnastkowo) nową wartość. W pewnych komórkach, zwłaszcza związanych z wyświetlaniem obrazu, XLF wymienia zawartość, aby być widocznym. Monitor wszakże pokazuje wartości takie, jak przed wywołaniem XLF-a. Zmiany w tych komórkach uwidoczną się dopiero po powrocie do przerwanej XLF-em programu. Wglądownicę da się przesunąć, gdyby coś zasłaniała, klawiszami ^+, ^*, ^-, ^=. Klawisz HELP przypomina najważniejsze informacje.

TEKSTY ŹRÓDŁOWE PROGRAMÓW

Oprócz programów narzędziowych, czyli plików typu COM, zawarto na dyskietce (a może kasecie) pewną liczbę tekstów źródłowych. Można je ogólnie podzielić na dwie kategorie: zestawy standardowych procedur i przykłady kompletnych programów, korzystających z tych procedur.

1. Zestawy standardowych procedur

Szereg zagadnień takich, jak komunikacja z urządzeniami WE/WY, grafika PM (graczy i pocisków), generowanie dźwięków, można, raz opracowawszy, adaptować niemal bez zmian do kolejnych programów. Doświadczony programista ma zwykle w swych zasobach procedury na tyle uniwersalne, by nie trzeba było ich pisać za każdym razem od nowa. Początkującym proponujemy nasze pomysły takich rozwiązań.

Pliki zawierające owe procedury mają szereg wspólnych cech. Zarówno nazwy plików, jak i wszystkie etykiety w nich użyte zaczynają się od pojedynczego znaku podkreślenia. Unikanie takich nazw w swoich programach pozwoli zapobiec nieoczekiwanym konfliktom. Każda procedura opisana jest w tekście z podaniem przeznaczenia, parametrów wejściowych i wyjściowych. Przeważnie więc wystarczy obejrzeć plik z procedurami, nie trzeba już czytać poniższych opisów.

Procedury można połączyć ze swym programem na dwa sposoby: poprzez użycie pseudorozkazu ICL, który powoduje dołączenie wskazanego pliku podczas asemblacji lub poprzez fizyczne "wklejenie" ich do tekstu programu przy pomocy komendy **^Kr** edytora. Ta druga metoda zalecana jest zwłaszcza dla użytkowników magnetofonu.

Nie sposób oczywiście przewidzieć wszystkich sytuacji i potrzeb, przeto prawdziwie uniwersalne procedury po prostu nie istnieją. Dla specjalnych zastosowań trzeba być może zmodyfikować ich tekst lub napisać nowe, wzorując się tylko na niektórych rozwiązaniach. Autorzy ani wydawca nie zgłaszają żadnych zastrzeżeń co do wykorzystania przedstawionych tekstów programów.

1.1. _IO

Plik _IO.ASM zawiera procedury komunikacji z urządzeniami WE/WY oraz prostej konwersji danych. Te z nich, które korzystają ze standardowych bloków IOCB (kanałów), wymagają podania w rejestrze X numeru IOCB-u pomnożonego przez 16. Niektóre procedury używają słowa na stronie zerowej. Trzeba je wskazać poprzez zadeklarowanie etykiety _ioz0 przed miejscem dołączenia procedur.

1.1.1. _open

Procedura ta otwiera kanał, wiążąc go logicznie z plikiem, którego nazwa znajduje się pod adresem zawartym w rejestrach A i Y. Znacznik przeniesienia C w chwili wywołania _open wskazuje na planowany kierunek przepływu danych: C=0 oznacza odczyt, zaś C=1 – zapis. Rejestr X określa nr IOCB. Po powrocie z procedury znacznik N wskazuje, czy operacja powiodła się (N=0 oznacza sukces), a rejestr Y zawiera numer ewentualnego błędu.

1.1.2. _close

Zamyka wskazany kanał. Rejestr X określa nr IOCB. Po powrocie znacznik N=0 oznacza sukces, w przeciwnym razie rejestr Y zawiera kod błędu.

1.1.3. _settxt

Ustawia tryb tekstowy przesyłania danych dla wszystkich następnych operacji _read i _write.

1.1.4. _setbin

Ustawia tryb binarny przesyłania danych dla wszystkich następnych operacji _read i _write.

1.1.5. _read

Odczytuje dane z pliku związanego ze wskazanym kanałem. Należy przewidzieć obszar na pomieszczenie tych danych. Adres obszaru przekazuje się do procedury _read w rejestrach A i Y. Pierwsze dwa bajty tego obszaru powinny zawierać długość jego pozostałej części. W niej zostaną umieszczone wczytane dane, a ich faktyczny rozmiar wpisany zostanie do początkowych dwóch bajtów obszaru. Rejestr X określa nr IOCB. Po powrocie N=0 wskazuje na poprawne wykonanie operacji, w przeciwnym razie rejestr Y zawiera numer błędu.

1.1.6. _write

Zapisuje dane na plik związany ze wskazanym kanałem. Dane do przesłania winny być poprzedzone słowem określającym ich długość, którego adres przekazuje się do procedury w rejestrach A i Y. Rejestr X określa nr IOCB. Po powrocie N=0 wskazuje na poprawne wykonanie operacji, w przeciwnym razie rejestr Y zawiera numer błędu.

1.1.7. _getkey

Czeka na naciśnięcie klawisza i wraca z kodem ASCII tego znaku w akumulatorze. Rejestr X określa nr IOCB, który ma zostać użyty podczas tej operacji.

1.1.8. _putch

Wysyła na ekran (na pozycji kursora) znak dostarczony w akumulatorze. Ponieważ omija standardowy blok IOCB, można ją wywoływać z dowolną wartością w rejestrze X. Uwaga: znaki sterujące są wykonywane (a nie wyświetlane) w sposób właściwy dla urządzenia "E:", chyba, że poprzedzono je znakiem sterującym o kodzie 27.

1.1.9. _phex

Ta procedura pozwala przedstawić bajt w formacie szesnastkowym (dwóch cyfr, będących znakami ASCII). Adres obszaru, w którym mają zostać umieszczone cyfry należy dostarczyć na stronie zerowej w słowie _ioz0. Miejsce wewnątrz tego obszaru, gdzie powinna pojawić się ostatnia (najmłodsza) cyfra wskazuje się rejestrem Y. Bajt do konwersji przekazuje się w akumulatorze. Procedura sama zmniejsza rejestr Y tak, że w przypadku liczb wielobajtowych wypisują się "od tyłu" w wyniku kolejnych jej wywołań bez dodatkowych ingerencji w zawartość Y.

1.1.10. _upper

Służy do zamiany pojedynczej małej litery alfabetu na dużą. Znak do konwersji podaje się w akumulatorze, tam też zjawia się wynik. Jeśli podany znak nie jest małą literą, to nie ulegnie zmianie.

1.2. _PM

Plik _PM.ASM zawiera proste procedury służące do poruszania graczami i pociskami, obiekty te będziemy krótko nazywać PM. Procedury używają czterech bajtów na stronie zerowej. Trzeba je wskazać poprzez zadeklarowanie etykiety _pmz0 przed miejscem dołączenia procedur.

1.2.1. _pmon

Ta procedura powoduje przełączenie komputera w tryb wyświetlania obiektów PM. W akumulatorze należy podać starszy bajt adresu obszaru grafiki PM. Trzeba przy tym pamiętać, że trzy najmłodsze bity akumulatora muszą być zerami (inaczej mówiąc: adres obszaru musi być podzielny przez 2048). Znacznik C wskazuje na rozdzielczość obiektów w pionie: C=0 oznacza mniejszą rozdzielczość, tzw. "dwuliniową".

1.2.2. `_pmset`

Ustawia kolor i szerokość gracza. W akumulatorze podaje się numer gracza (0..3), w rejestrze X kod koloru (16 * nr barwy + jasność), a w rejestrze Y – szerokość gracza: 1 – normalna, 2 – podwójna, 4 – poczwórna.

1.2.3. `_pmshape`

Pozwala skopiować kształt gracza z obszaru definicji do obszaru grafiki PM. Numer gracza (0..3) podaje się w akumulatorze. Adres 16-bajtowej definicji gracza zawarty jest w rejestrach X i Y. Obraz gracza pojawi się w miejscu ustalonym w wyniku ostatniego użycia procedury `_pmXY`. Warto więc zauważyć, że można mieć wiele definicji jednego gracza i przełączać je w dowolnym tempie, co umożliwia animację. Nie ma natomiast możliwości wpływania na wygląd pocisków, które mają zawsze kształt kropki.

1.2.4. `_pmXY`

Procedura ta ustawia obiekt, którego numer podano w akumulatorze, na ekranie w miejscu o współrzędnych podanych w rejestrach X i Y. Numery 0..3 odnoszą się do graczy, zaś numery 4..7 – do pocisków.

1.2.5. `_pmoff`

Powoduje wyłączenie wyświetlania obiektów PM.

1.3. `_SOUND`

Plik `_SOUND.ASM` zawiera proste procedury służące do wytwarzania dźwięków za pomocą generatorów układu POKEY. Procedury używają dwóch bajtów na stronie zerowej. Trzeba je wskazać poprzez zadeklarowanie etykiety `_sndz0` przed miejscem dołączenia procedur.

1.3.1. _sound

Włącza generator, którego numer (od 0 do 3) podany jest w X, okres dźwięku (od 0 do 255) w Y, zaś barwa i głośność - w A, według wzoru $16 * \text{barwa} + \text{głośność}$. Barwa dźwięku musi być liczbą parzystą z zakresu 0..14.

1.3.2. _wait

Wstrzymuje działanie programu na czas podany w akumulatorze. Jednostką czasu jest jedno wyświetlenie obrazu w telewizorze, czyli w przypadku systemu PAL 1/50 s. Jak łatwo obliczyć, największa zwłoka (uzyskana z liczby 255), to około 5 sekund.

1.3.3. _musini

Ustawia stan początkowy dla kolejnych wywołań procedury _play. W akumulatorze podaje się głośność przyszłych dźwięków, w rejestrach X i Y - adres definicji muzyki. Definicję stanowi dowolny ciąg bajtów, w którym dźwięki dwóch oktaw oznaczone są znakami: C, D, E, F, G, A, H, c, d, e, f, g, a, h. Znak "#" umieszczony bezpośrednio po nutce podwyższa dźwięk o pół tonu. Dowolny znak o kodzie wyższym niż 127, np. 155 (End-Of-Line), oznacza koniec melodii. Nie wymienione wyżej znaki interpretowane są jako pauzy. Procedura ta wycisza również wszystkie dźwięki.

1.3.4. _play

Gra kolejny dźwięk z ciągu bajtów określonego ostatnim wywołaniem procedury _musini. Procedura nie wymaga podawania parametrów w rejestrach. W chwili powrotu z procedury znacznik C=1 oznacza, że zagrano już wszystkie dźwięki. Procedura odtwarza dźwięki, używając generatorów 0 i 1 układu POKEY, pozostałe dwa można wykorzystać według uznania.

2. Programy przykładowe

Przykładowe programy powstały dla zademonstrowania sposobu użycia zestawów standardowych procedur. Nie mają ambicji stać się wzorcem do naśladowania. Przypuszczalnie każdy(a) z Was bez trudu napisze znacznie lepsze. Programy te umieszczają się w pamięci pod adresem \$8800 i od tego adresu należy rozpocząć ich wykonywanie.

D Dyskietka dystrybucyjna zawiera w związku z tym plik konfiguracyjny "QA.SET", który ustawia adres **Memhi**, czyli początek wolnej pamięci, oraz adres **Run**, to jest początek logiczny programu, oba na \$8800.

C W związku z tym użytkownik zobowiązany jest ustawić adresy **Memhi** oraz **Run** w podmenu **Setup**, oba na \$8800, najlepiej przed rozpoczęciem wczytywania przykładowych programów. Pierwszy definiuje początek wolnej pamięci, zaś drugi – początek logiczny programu.

2.1. TRANS

Jest to przykład kopiera wyposażonego w możliwość transkodowania (translacji) plików z formatu stosowanego w pakiecie JBW Assembler do "klasycznego" pliku ASCII, zdatnego do przetwarzania przez dowolny edytor, w tym głównie QA. Program ten wyświetla informację o używanej pamięci oraz literowe menu, które umożliwia:

- C – wczytanie pliku do pamięci kopiera,
- Z – zapis pliku z pamięci na urządzeniu zewnętrznym,
- T – translacja pliku z pamięci z formatu "JBW Assembler" na ASCII z zapisem na urządzeniu,
- K – opuszczenie programu.

Aby uruchomić program bezpośrednio z tekstu, trzeba wczytać QA, ustawić (**Setup**) adresy **Run** i **Memhi** oba na \$8800, wgrać plik TRANS.ASM do edytora, i wykonać funkcję **Assembly**, a następnie **Run**.

Można też wyprodukować plik binarny, który później da się wykorzystywać niezależnie od QA. W tym celu trzeba w tekście programu TRANS dokonać kilku zmian. Na samym końcu pliku, tuż przed końcowym END należy dodać dwie instrukcje:

```
ORG    $2E0
DTA    A(MAIN)
```

One tam są, lecz ich nie ma, bo stanowią część komentarza. Przywrócimy ich obecność, usuwając gwiazdeczki. Dzięki temu DOS lub COS dowie się, od jakiego adresu uruchomić ten program. W początkowej części pliku rozkaz OPT definiuje parametry asemblacji. Trzeba zmienić fragment dotyczący generowania kodu z code_mem na code_dsk, co spowoduje podczas asemblacji wystanie wyniku na plik zewnętrzny. Można też zmienić (choć nie jest niezbędne) umieszczoną tuż za deklaracjami instrukcję ORG \$8800 na inną, w zależności od używanego systemu, zwiększając dostępną dla kopiera pamięć, np. na ORG \$2400.

- D** Po opuszczeniu edytora dobrze jest poprawić jeszcze nazwę pliku wyjściowego (**File: Obj**) na TRANS.COM, bo będzie to samodzielny program (komenda zewnętrzna DOS-u). Teraz w wyniku asemblacji powstanie na dysku gotowy program.
- C** Nie jest możliwe zapisywanie kodu na taśmie z programu, który zawiera rozkazy ICL. Trzeba więc usunąć ten rozkaz, dołączając zamiast niego plik wymieniony w tym rozkazie (należy zastosować polecenie **^Kr** edytora). Po opuszczeniu edytora polecenie asemblacji spowoduje zapisanie na taśmie pliku z programem TRANS, który może być odtąd traktowany jako komenda zewnętrzna COS-u. Dobrze jest taki plik wczytać i znów zapisać programem NCOPY, aby skrócić przerwy między rekordami i dodać nazwę.

2.2. DEMO

Przykład wykorzystania procedur bibliotecznych _PM i _SOUND. Proponujemy zwrócić uwagę na sposób wykorzystania procedury _play, pozwalającej na jednoczesną animację obiektów PM i odtwarzanie

muzyki. Aby uruchomić program bezpośrednio z tekstu, trzeba wczytać QA, ustawić (**Setup**) adresy **Run** i **Memhi** oba na \$8800, wgrać plik DEMO.ASM do edytora, wykonać funkcję **Assembly**, a potem **Run**.

Wszystkich zainteresowanych poważnie programowaniem ATARI w języku asemblera zachęcam do sięgania po miesięcznik "TAJEMNICE ATARI", w którym ta miła sercu tematyka znalazła swój stały kącik.

Pozdrawiam Was serdecznie, zapaleni programiści niskiego poziomu (niskiego poziomu, lecz wysokiego kunsztu). Życzę Wam wielu radosnych dni i nocy spędzonych z przedstawionymi tu narzędziami.

autor