

**WARSZAWSKA  
WYŻSZA SZKOŁA INFORMATYKI**

---

**PRACA DYPLOMOWA  
WYŻSZE STUDIA ZAWODOWE**

**Maciej GRZESZCZUK**

Numer albumu 2229

**Wieloprocessowy system operacyjny  
dla komputerów Atari XL/XE**

**Promotor:  
mgr inż. Dariusz MAJKA**

Praca spełnia wymagania stawiane pracom  
inżynierskim na studiach inżynierskich.

Dnia ..... 2006r.

---

W A R S Z A W A 2006

## SPIS TREŚCI

SPIS TREŚCI .....	2
1. WSTĘP .....	4
2. WYKAZ UŻYTYCH SKRÓTÓW I WAŻNIEJSZYCH OZNACZEŃ .....	7
3. OPIS PLATFORMY ATARI XL/XE .....	11
3.1. HISTORIA .....	11
3.2. ARCHITEKTURA SPRZĘTOWA .....	16
3.2.1. CPU .....	16
3.2.2. Pamięć operacyjna .....	18
3.2.3. Przerwania .....	20
3.2.4. Procesor graficzny ANTIC .....	21
3.2.5. POKEY .....	22
3.3. URZĄDZENIA ZEWNĘTRZNE .....	22
3.4. ATARI OS .....	24
3.4.1. Podsystem wejścia/wyjścia .....	25
3.4.2. Procedury odczytu wstępnego .....	27
3.5. ROZWÓJ PLATFORMY .....	29
3.5.1. Rozwój oprogramowania .....	29
3.5.2. Modyfikacje sprzętowe .....	30
4. ŚRODOWISKO PRACY .....	34
4.1. OPROGRAMOWANIE NA ATARI .....	34
4.2. OPROGRAMOWANIE NA PC .....	35
4.3. SPRZĘT .....	35
4.4. ORGANIZACJA PLIKÓW ŹRÓDŁOWYCH .....	35
4.5. TWORZENIE PLIKÓW WYNIKOWYCH .....	36
4.5.1. Pliki systemu operacyjnego .....	36
4.5.2. Pliki procesów .....	38
5. PODSTAWOWE ELEMENTY SYSTEMU OPERACYJNEGO .....	39
5.1. PROCES .....	39
5.1.1. Strona bazowa procesu .....	39
5.1.2. Stos procesora, strona zerowa i przestrzeń IOCB. ....	40
5.2. ZARZĄDZANIE PAMIĘCIĄ .....	41
5.2.1. Obsługa dodatkowej pamięci komputera .....	41
5.2.2. Przydział pamięci dla procesów .....	41
5.2.3. Sztuczne tryby adresowania .....	45
5.3. ZARZĄDZANIE PROCESAMI .....	46
5.3.1. Cykl życia i stany procesów .....	47

5.3.2.	Uruchamianie procesów.....	49
5.3.3.	Planista długoterminowy.....	49
5.3.4.	Planista krótkoterminowy .....	50
5.3.5.	Przerwania systemowe w środowisku wielopprocesowym .....	55
5.3.6.	Semaforzy .....	57
5.3.7.	Terminacja procesu.....	59
5.4.	STEROWNIKI URZĄDZEŃ .....	60
5.4.1.	Konsole wirtualne.....	60
5.4.2.	Pamięci masowe i urządzenia szeregowy .....	65
5.4.3.	Urządzenia sieciowe .....	65
5.5.	KOMUNIKACJA MIĘDZYPROCESOWA .....	66
5.5.1.	Przekazywanie parametrów do procesu.....	66
5.5.2.	Sygnały.....	67
6.	PRZYKŁADOWE APLIKACJE.....	71
6.1.	APLIKACJE SYSTEMOWE .....	71
6.1.1.	shell.....	71
6.1.2.	ps.....	73
6.1.3.	kill.....	74
6.1.4.	Monitor systemowy .....	75
6.2.	PROGRAMY UŻYTKOWNIKA .....	76
6.2.1.	write .....	76
7.	PODSUMOWANIE .....	79
7.1.	ROZWÓJ PROJEKTU .....	81
8.	ZAŁĄCZNIKI .....	82
8.1.	INSTRUKCJA UŻYTKOWNIKA .....	82
8.1.1.	Organizacja płyty CD .....	82
8.1.2.	Instalacja emulatora Atari .....	83
8.1.3.	Praca z XEUX .....	83
8.2.	INSTRUKCJA PROGRAMISTY.....	84
8.2.1.	Mapa pamięci .....	84
8.2.2.	Dostęp do urządzeń.....	86
8.2.3.	Asembler 6502.....	87
8.2.4.	Tworzenie pliku binarnego w formacie relokalnym XEUX .....	91
8.2.5.	Kody błędów .....	94
	WYKAZ RYSUNKÓW.....	96
	WYKAZ TABEL.....	97
	BIBLIOGRAFIA .....	98

## 1. WSTĘP

System operacyjny to program będący pośrednikiem pomiędzy użytkownikiem komputera a zasobami sprzętowymi – jak procesor, pamięć i urządzenia wejścia-wyjścia – składającymi się na komputer<sup>1</sup>. Ma on za zadanie tworzyć środowisko, w którym użytkownik może uruchamiać własne programy w sposób efektywny i jednocześnie wygodny. Realizuje również dystrybucję dostępnych zasobów systemu komputerowego, przydzielając je użytkownikom oraz uruchamianym przez nich programom wtedy, gdy są one niezbędne do realizacji ich celów. Inna definicja systemu operacyjnego mówi, że „jest to program, który działa w komputerze nieustannie (...), podczas gdy wszystkie inne są programami użytkowymi”<sup>2</sup>.

W latach czterdziestych i pięćdziesiątych nie stosowano systemów operacyjnych – programy użytkowników były ręcznie ładowane do pamięci komputera za pomocą urządzeń wejściowych (np. czytników kart perforowanych), a wystąpienie błędu zatrzymywało ich wykonywanie. Określenie przyczyny błędu wymagało, aby użytkownik samodzielnie prześledził program, posiłkując się analizą rejestrów procesora i pamięci zwróconych w momencie wystąpienia błędu. W przypadku prawidłowego zakończenia działania, wynik przeważnie zwracany był jako wydruk z drukarki.<sup>3</sup>

W celu podniesienia efektywności, w połowie lat pięćdziesiątych General Motors zastosowało pierwszy system wsadowy, przeznaczony do współpracy z komputerem IBM 701<sup>4</sup>. Zadania do wykonania były umieszczane przez użytkownika w oddanym do dyspozycji systemu czytniku kart perforowanych bądź pamięci taśmowych, natomiast wyborem spośród nich zadania do uruchomienia oraz jego wczytaniem do pamięci operacyjnej zajmował się rezydentny program zwany monitorem. Po zakończeniu pracy programu kontrola zwracana była do monitora, który wybierał i uruchamiał kolejny z oczekujących programów.

Korzeni powstania systemów operacyjnych podziału czasu należy szukać w Massachusetts Institute of Technology (MIT), gdzie w roku 1961 na maszynach IBM 709

---

<sup>1</sup> Na podstawie: A. Silberschatz, P. B. Galvin, *Podstawy systemów operacyjnych*, Warszawa 2002, Wydawnictwa Naukowo-Techniczne

<sup>2</sup> A. Silberschatz, P. B. Galvin, *Podstawy systemów operacyjnych*, Warszawa 2002, Wydawnictwa Naukowo-Techniczne

<sup>3</sup> Na podstawie: W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic

<sup>4</sup> Na podstawie: W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic

wdrożony został Compatible Time Sharing System (CTSS)<sup>5</sup>. Prosty system przydziału pamięci – każdy proces rozpoczynał się dokładnie od tego samego miejsca – rozwiązywał problem relokacji kodu, przełączenia procesów odbywały się w takt występującego co 0,2 sekundy przerwania. Dane wywłaszczanego programu, wraz z jego kodem, przechowywane były na dysku. Część zespołu pracująca nad CTSS niespełna 10 lat później, bo w 1970 roku, uruchomiła w zakładach Bell Labs pierwszą wersję systemu UNIX, przeznaczoną dla komputerów DEC PDP-7. Dalszy rozwój systemów podziału czasu spowodował szybkie upowszechnienie się ich na wszystkie używane obecnie platformy sprzętowe.

Komputery domowe Atari pojawiły się w końcu lat siedemdziesiątych ubiegłego wieku. Grupą docelową odbiorców były rodziny amerykańskie, którym komputer miał służyć do rozrywki, nauki, czy prowadzenia średniej wielkości działalności gospodarczej. Dobrze przemyślana konstrukcja i duże możliwości sprawiły, że rynek ośmiobitowych Atari na zachodzie miał się dobrze do końca lat osiemdziesiątych, mimo ciągle umacniających swoją pozycję komputerów PC. W Europie wschodniej na okres drugiej młodości tych komputerów przypadają lata 1992-1994.

Nie każdy jednak wie, że komputery Atari mogą z łatwością zostać połączone w sieć, w której jednostka pełniąca rolę serwera automatycznie zaaplikuje i uruchomi odpowiednie oprogramowanie klienckie podłączonym do niej maszynach, przy czym klienci nie muszą w tym celu uruchamiać żadnego dodatkowego oprogramowania ponad procedury umieszczone w ich pamięci ROM przez programistów z firmy Atari w końcu lat 70-tych ubiegłego wieku. Doskonale zaprojektowany system obsługi rozszerzeń pamięci wprowadzony w komputerach serii XE na początku lat 80-tych umożliwia powiększanie dostępnej pamięci do rozmiaru 1MB, przy zachowaniu pełnej kompatybilności z liczącymi sobie często ponad 20 lat aplikacjami mogącymi pamięć taką wykorzystywać. Urządzenia takie jak kontrolery SCSI czy IDE, pozwalające na podłączenie twardych dysków, napędów CD czy czytnika kart Compact Flash, zbudowane według wytycznych projektantów komputera, rejestrują się w systemie i działają sprawnie bez potrzeby wczytywania jakichkolwiek sterowników. Mimo wielu niezaprzeczalnych zalet, nigdy jednakże ośmiobitowe komputery Atari nie doczekały się powstania wieloprotocowego systemu operacyjnego z podziałem czasu.

Celem niniejszej pracy jest zapoznanie czytelnika z ośmiobitowym komputerem Atari, występującym w odmiennej od pamiętanej roli domowego komputera, w którym najczęściej używanym urządzeniem zewnętrznym był wysłużony joystick. Zadaniem stojącym przed autorem jest potwierdzenie, iż stworzenie pełnosprawnego wieloprotocowego systemu

---

<sup>5</sup> Na podstawie: W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic

operacyjnego dla niezmodyfikowanego komputera Atari XL/XE było w przeszłości i jest obecnie możliwe, a fakt, iż do dnia dzisiejszego taki nie powstał, tłumaczyć można niskim zainteresowaniem użytkowników, a co za tym idzie nieopłacalnością projektu z punktu widzenia firm produkujących oprogramowanie, nie natomiast brakiem technicznych możliwości.

**Rozdział 2** zapoznaje czytelnika ze skrótami i ważniejszymi oznaczeniami zastosowanymi w pracy.

**Rozdział 3** prezentuje historię ośmiobitowej linii komputerów Atari i ich rozwój zarówno w ramach działalności producenta, jak i będący efektem pracy grup fascynatów z różnych regionów świata. Opisane są również zwięzłe elementy architektury sprzętowej, o których wiedza da pogląd na temat punktu startowego projektu oraz mechanizmów użytecznych podczas realizacji zadania.

**Rozdział 4** niniejszego opracowania opisuje środowisko pracy stworzone na potrzeby projektu.

**Rozdział 5** to szczegółowy opis wszystkich wdrożonych elementów stworzonego systemu operacyjnego. Jego zawartość podzielona jest na kilka istotnych funkcji, jak zarządzanie procesami, zarządzanie pamięcią, komunikacja międzyprocesowa, czy obsługa urządzeń wejścia-wyjścia. Funkcjonowanie krytycznych elementów jądra systemu przedstawione jest w formie czytelnych diagramów blokowych, a sposób użycia funkcji systemowych jest dogłębnie opisany oraz opatrzone krótkimi wstawkami kodu demonstracyjnego.

**Rozdział 6** przedstawia przykładowe aplikacje narzędziowe i demonstracyjne, jakie zostały napisane dla utworzonego systemu.

**Rozdział 7** jest podsumowaniem efektów pracy. Informacje w nim zawarte przedstawiają zrealizowane cele i w formie zwięzłych cech charakteryzują powstały produkt. W końcowej jego części zarysowany jest potencjalny kierunek rozwoju powstałego produktu.

**Rozdział 8** zawiera załączniki, a pośród nich m.in.: mapę pamięci nowego systemu, instrukcję użytkownika, jak również istotne informacje dotyczące interfejsu programisty, łącznie z krótką charakterystyką języka assemblera procesora 6502 i kodami błędów zwracanymi przez procedury nowego systemu. W rozdziale szczegółowo opisana jest również budowa pliku binarnego nowego formatu, wraz ze sposobem jego przygotowywania.

## 2. WYKAZ UŻYTYCH SKRÓTÓW I WAŻNIEJSZYCH OZNACZEŃ

<b>CPU</b>	- <i>Central Processing Unit</i> , inaczej procesor.
<b>6502</b>	- 8 bitowy mikroprocesor zaprojektowany przez MOS Technology w roku 1975 <sup>6</sup> , wykorzystywany m.in. przez Atari dla komputerów domowych serii 400/800 oraz XL/XE – w tych ostatnich wykorzystywana jest uzupełniona o dodatkowy sygnał HALT wersja oznaczana 6502C (SALLY).
<b>RAM</b>	- <i>Random Access Memory</i> . Pamięć operacyjna komputera.
<b>ROM</b>	- <i>Read Only Memory</i> . Pamięć tylko do odczytu, której zawartość jest stała.
<b>PBI</b>	- <i>Parallel Bus Interface</i> . Interfejs równoległy w komputerach Atari XL.
<b>ECI</b>	- <i>Enhanced Cartridge Interface</i> . Interfejs równoległy w komputerach Atari serii XE, nieznacznie różniący się od PBI.
<b>ANTIC</b>	- <i>Alpha-Numeric Television Interface Controller</i> . Procesor graficzny, odpowiedzialny za generację obrazu w ośmiobitowych Atari.
<b>SIO</b>	- <i>Serial Input/Output</i> . Gniazdo interfejsu szeregowego w komputerach Atari XL/XE, również nazwa podsystemu niskopoziomowych procedur blokowego dostępu do urządzeń szeregowych <sup>7</sup> .
<b>XMS</b>	- <i>eXtended Memory Specification</i> . Pamięć dodatkowa, podłączana w formie 16kB banków w obszar adresowy 0x4000-0x7FFF.
<b>FIFO</b>	- <i>First-In-First-Out</i> . Kolejka typu pierwsze na wejściu, pierwsze na wyjściu.
<b>POKEY</b>	- <i>Pot-Keyboard Integrated Circuit</i> . Układ wejścia/wyjścia, odpowiedzialny w Atari za generację dźwięku, kontrolę interfejsu szeregowego (SIO), obsługę manipulatorów analogowych (wiosełek, zwanych również <i>paddle</i> ) oraz klawiatury <sup>8</sup> .

---

<sup>6</sup> Na podstawie [http://en.wikipedia.org/wiki/MOS\\_Technology\\_6502](http://en.wikipedia.org/wiki/MOS_Technology_6502)

<sup>7</sup> Na podstawie K. M. Kokoszkiewicz, *SIO*, atariki, 2005, <http://atariki.krap.pl/index.php/SIO>

<sup>8</sup> Na podstawie K. M. Kokoszkiewicz, *POKEY*, atariki, 2005, <http://atariki.krap.pl/index.php/POKEY>

<b>PIA</b>	- <i>Peripheral Interface Adapter</i> . Układ MOS 6520, w komputerach Atari obsługuje porty joysticków, a w serii XL/XE steruje również kontrolerem pamięci <sup>9</sup> .
<b>IRQ</b>	- <i>Interrupt ReQuest</i> . Przerwanie maskowalne procesora 6502.
<b>NMI</b>	- <i>Non-Maskable Interrupt</i> . Przerwanie niemaskowalne procesora 6502.
<b>DLI</b>	- <i>Display List Interrupt</i> . Jedno z przerwania niemaskowalnych generowanych przez procesor ANTIC po napotkaniu stosownej instrukcji.
<b>VBL</b>	- <i>Vertical BLank interrupt</i> . Przerwanie generowane przez ANTIC po zakończeniu generowania obrazu, w momencie wygaszenia plamki <sup>10</sup> .
<b>GTIA</b>	- <i>Graphics Television Interface Adapter</i> . Układ generujący sygnał wizyjny na podstawie danych od procesora graficznego ANTIC.
<b>LSB</b>	- <i>Less Significant Byte</i> . Mniej znaczący bajt dwubajtowego słowa.
<b>MSB</b>	- <i>Most Significant Byte</i> . Bardziej znaczący bajt dwubajtowego słowa.
<b>DOS</b>	- <i>Disk Operating System</i> . Wbrew nazwie, nie jest to system operacyjny, a zestaw procedur obsługi stacji dysków oraz dysków twardych doczytywanych do systemu operacyjnego podczas inicjalizacji systemu.
<b>UART</b>	- <i>Universal Asynchronous Receiver Transmitter</i> . Obwód zintegrowany używany do asynchronicznego przekazywania i odbierania informacji poprzez port szeregowy <sup>11</sup> .
<b>CIO</b>	- <i>Central Input/Output</i> . Podsystem wejścia/wyjścia w Atari OS.
<b>IOCB</b>	- <i>Input/Output Control Block</i> . Blok kontrolny wejścia/wyjścia, opisujący jeden kanał komunikacyjny podsystemu CIO.
<b>HATABS</b>	- <i>Hardware Address Table</i> . Tablica deskryptorów urządzeń, część podsystemu CIO.
<b>Boot</b>	- Procedura odczytu wstępnego, wykonywana podczas startu systemu z kasety lub dysku.

---

<sup>9</sup> Na podstawie K. M. Kokoszkiwicz, *PIA*, atariki, 2005, <http://atariki.krap.pl/index.php/PIA>

<sup>10</sup> Na podstawie K. M. Kokoszkiwicz, *VBL*, atariki.krap.pl, 2005, <http://atariki.krap.pl/index.php/VBL>

<sup>11</sup> <http://pl.wikipedia.org/wiki/UART>



<b>PAL</b>	- <i>Phase-Alternating Line</i> . Europejski system kodowania koloru w transmisji telewizyjnej. Nieformalnie używany jako nazwa systemu generacji obrazu, który składa się z 625 linii przy częstotliwości odświeżania 50Hz (25 półobrazów na sekundę).
<b>NTSC</b>	- <i>National Television Systems Committee</i> . Amerykański system generacji obrazu telewizyjnego, w którym obraz składa się z 525 linii przy częstotliwości odświeżania 59.94Hz <sup>12</sup> (29.97 półobrazów na sekundę).
<b>PC</b>	- <i>Program Counter</i> . Szesnastobitowy rejestr procesora, wskazujący w pamięci instrukcję, która powinna być wykonana jako następna.
<b>ALU</b>	- <i>Arithmetic Logic Unit</i> . Jednostka wewnątrz procesora, której przeznaczeniem jest wykonywanie operacji arytmetycznych oraz logicznych.
<b>NB</b>	- Naturalny kod binarny. Podstawowy sposób bitowej reprezentacji liczb całkowitych bez znaku.
<b>U2</b>	- Kod uzupełnień do dwóch. Obecnie najpopularniejszy sposób zapisu bitowego liczb całkowitych ze znakiem.
<b>BCD</b>	- <i>Binary Coded Decimal</i> . Sposób binarnego zapisu liczb dziesiętnych, w którym każdej cyfrze liczby odpowiadają 4 bity, przy czym bity te mogą kodować jedynie liczby od 0 do 9. Przykładowo, liczba 94 będzie zapisana w BCD jako 10010100, a reprezentacja tego zapisu w systemie szesnastkowym to \$94.
<b>TTY</b>	- Terminal tekstowy. W XEUX konsola wirtualna, tworzona przez osobną przestrzeń na pamięć ekranu i odrębny bufor klawiatury.
<b>XXCIO</b>	- <i>Xeux eXtended Central Input/Output</i> . Dostosowany do wymagań systemu wielopprocesowego podsystem CIO, wdrożony w XEUX.
<b>MAE</b>	- Pakiet makroassembler+debuger autorstwa J. Harrisa; również trzyliterowe rozszerzenie wyróżniające pliki źródłowe napisane w składni zgodnej z wyżej wymienionym pakietem.
<b>ATASCII</b>	- Podzbiór zestawu znaków ASCII, wykorzystywany w ośmiobitowych komputerach Atari. Zawiera 128 znaków (w zakresie od 0 do 127), górne wartości kodów odpowiadają negatywom znaków z pierwszej połowy.

---

<sup>12</sup> Źródło: <http://en.wikipedia.org/wiki/NTSC>

- UNIX** - Potocznie nazwa ta jest używana do określania rodziny systemów operacyjnych zgodnych z zestandaryzowanymi normami (POSIX, Single UNIX Specification). Nazwa UNIX jest znakiem towarowym zastrzeżonym przez konsorcjum The Open Group<sup>13</sup>.
- FIXUP** - Dwubajtowy indeks miejsca w kodzie programu, które należy zmodyfikować dodając adres strony pamięci, na której ulokowany będzie dany program. Tablica indeksów tych dołączana jest do pliku binarnego celem umożliwienia właściwej relokacji pliku po jego wczytaniu.
- ZOMBIE** - Stan procesu po zakończeniu działania, kiedy uwolnione są już zasoby procesu, ale pozostaje wpis jego dotyczący na stronie bazowej. Wpis taki umożliwiałby procesowi-rodzicowi odczytanie statusu zakończenia potomka, jednak z jakichś przyczyn rodzic nie pobrał jeszcze tych informacji.

---

<sup>13</sup> Źródło: [http://www.unix.org/what\\_is\\_unix.html](http://www.unix.org/what_is_unix.html)

### 3. OPIS PLATFORMY ATARI XL/XE

#### 3.1. HISTORIA

W roku 1978 firma Atari wprowadziła do swojej oferty dwa pierwsze komputery domowe, Atari 400 i Atari 800. Konstrukcje te, oparte na bazie procesora 6502 firmy MOS Technology, różniły się szczegółami wynikającymi z marketingowego pozycjonowania obu urządzeń. Atari 800 – znany w korporacji pod kryptonimem Colleen<sup>14</sup> – został zaprojektowany jako półprofesjonalny komputer domowy, wyposażony w wygodną klawiaturę, był również łatwo rozszerzalny – użytkownik miał możliwość powiększenia pamięci RAM (do maksymalnej wielkości 48kB) bądź aktualizacji systemu operacyjnego wykonując prostą czynność wymiany bądź dołożenia estetycznego modułu dostępnego w detalicznej sprzedaży. Kryjący się pod wewnętrzną nazwą Candy komputer Atari 400 skierowano do młodszych bądź mniej wymagających użytkowników, głównie jako komputer do gier i rozrywki – wyposażony był w niewygodną, membranową klawiaturę, usunięto wyjście monitorowe, dodatkowy port cartridgea i możliwość rozszerzania pamięci lub aktualizacji systemu operacyjnego bez udziału serwisu.

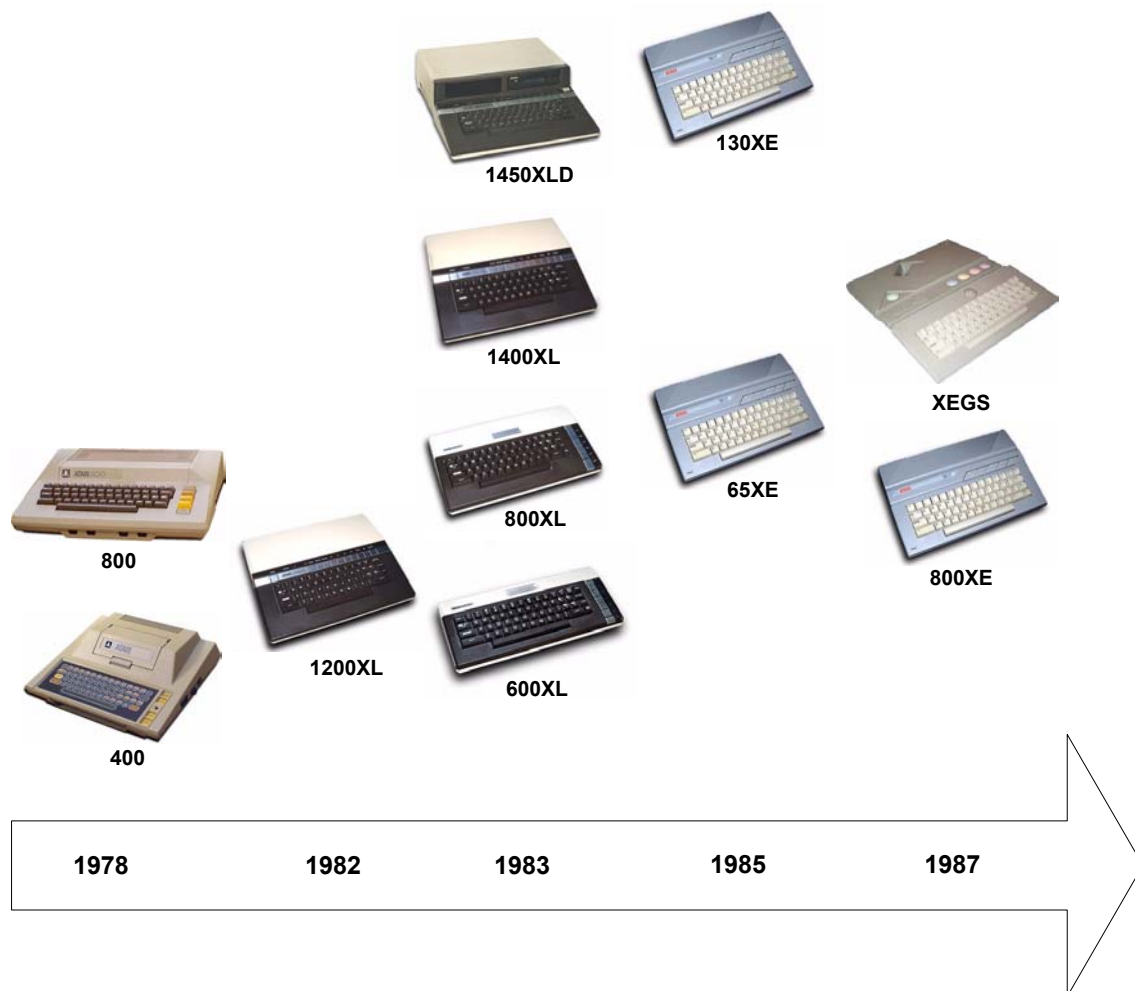
Kontynuację linii 400/800 stanowiły od 1982 maszyny serii XL - istotną zmianą było dodanie interfejsu PBI umożliwiającego podłączenie dodatkowych urządzeń bezpośrednio do szyny równoległej komputera. Pociągnęło to za sobą rozbudowę systemu operacyjnego o dodatkowe procedury obsługi tych urządzeń. Zredukowano liczbę portów joysticków do dwóch, rozszerzono również pamięć ROM do 24kB (z czego 8kB przeznaczono na wbudowany interpreter języka BASIC). Fabrycznie zainstalowane (z wyjątkiem modelu 600XL) 64kB pamięci RAM pokrywa całą dostępną przestrzeń adresową procesora, łącznie z obszarami zajętyymi przez pamięć ROM – projektanci przewidzieli możliwość programowego odłączania segmentów pamięci ROM celem zastąpienia jej własnym kodem umieszczonym w pamięci operacyjnej. Dzięki temu użytkownik może zastąpić kod wbudowanego systemu operacyjnego własnymi, ulepszonymi procedurami, bądź też wykorzystać całą dostępną pamięć dla swojego oprogramowania. Tą możliwość wykorzystuje pakiet „Translator”, umożliwiający uruchamianie na komputerach serii XL/XE oprogramowanie dla komputerów serii 400/800, które korzysta z bezpośrednich skoków do systemu, z pominięciem zdefiniowanych wektorów wejścia – pakiet ten podmienia w całości system operacyjny,

---

<sup>14</sup> Na podstawie [http://en.wikipedia.org/wiki/Atari\\_800#The\\_early\\_machines:\\_400\\_and\\_800](http://en.wikipedia.org/wiki/Atari_800#The_early_machines:_400_and_800)

wyłączając ROM XL/XE<sup>15</sup>.

Największą popularność w Polsce zdobyły komputery Atari XE, głównie dzięki dystrybucji poprzez sieć Pewex. Poza zmianą linii wzorniczej – estetyczne szare obudowy znane również z 16-bitowych komputerów Atari ST – oraz podwojoną ilością pamięci RAM w modelu 130XE nie różniły się funkcjonalnie od komputerów serii XL.



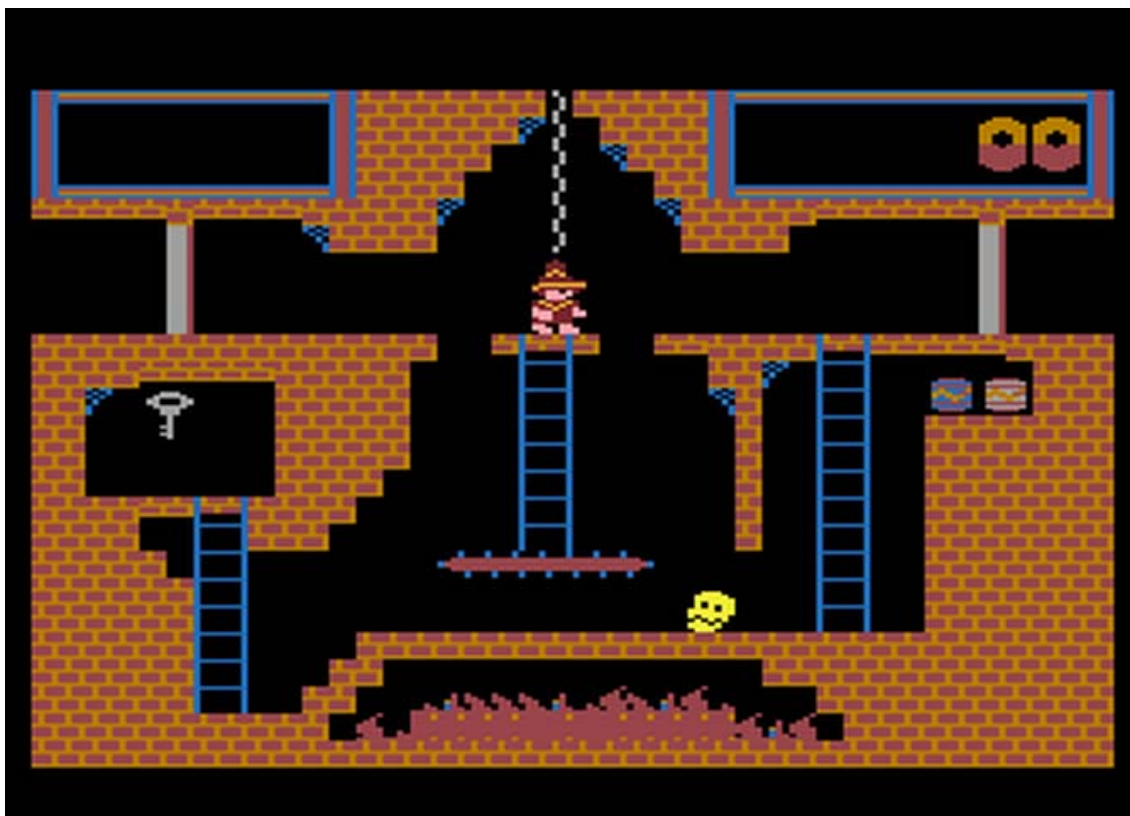
Rys. 3.1. Rozwój komputerów domowych Atari

Źródło: opracowanie własne, na podstawie <http://atariki.krap.pl/>, <http://www.atari-museum.de/> i [http://en.wikipedia.org/wiki/Atari\\_800](http://en.wikipedia.org/wiki/Atari_800)

Podstawowym zastosowaniem komputerów serii XL/XE była rozrywka – duża baza gier komputerowych zapewniała nieprzespane noce licznej grupie użytkowników. Często całe rodziny, bez względu na wiek, bawiły się dobrze przy co bardziej popularnych tytułach, jak

<sup>15</sup> Na podstawie <http://atariki.krap.pl/index.php/XL-Fix>

River Raid, Montezuma's Revenge, Blue Max, Zorro, Laser Hawk czy Super Cobra. Gęsto rozsiadane po terenie kraju „studia komputerowe” oraz cotygodniowe giełdy oferowały szeroki asortyment gier, programom użytkowym poświęcając przeważnie mniej uwagi.



Rys. 3.2. *Montezuma's Revenge* autorstwa R. Jaegera

Źródło: kopia ekranu gry

Część winy za taki stan rzeczy można przypisać faktowi, iż praca z programami użytkowymi uruchamianymi na komputerze do którego w charakterze pamięci masowej podłączony był magnetofon, nie należała do rzeczy przyjemnych. Barierą był też fakt, iż większość oprogramowania użytkowego była wyłącznie w języku angielskim, z nielicznymi wyjątkami w języku niemieckim. Przeważnie były to też pirackie kopie, pozbawione dokumentacji, bądź ze sporządzonymi na maszynie do pisanie i wielokrotnie kserowanymi, czterostronicowymi „opisami”, w których autorzy opisywali tylko te funkcje, które im samym udało się odkryć eksperymentując. Poradniki dotyczące programowania nie były znacząco lepszej jakości – nieautoryzowane, nieudolne tłumaczenia zachodnich wydawnictw, wydawane w formie nieczytelnych odbitek kserograficznych, ograniczały kontakt z programowaniem przeważnie do podstaw wbudowanego języka BASIC.

Sytuacja zmieniła się na początku lat 90-tych, kiedy cena stacji dysków stała się bardziej przystępna, a jednocześnie działalność rozpoczęły polskie firmy jak *MIRAGE* czy *Laboratorium Komputerowe Avalon*, wydające legalne oprogramowanie użytkowe w języku polskim i z myślą o polskim użytkowniku. Pojawiły się regularne polskojęzyczne wydawnictwa, główne tytuły to *Atari Magazyn*, *Świat Atari* i *Tajemnice Atari*, które popularyzowały użytkowe zastosowania komputerów oraz wprowadzały czytelników w arka programowania. W okresie tym pojawiła się także tzw. „scena”, czyli nieformalna grupa ludzi skupiona wokół tematyki Atari, przeważnie składająca się z mniejszych jednostek zwanych grupami. Grupy zrzeszały grafików, muzyków, koderów, również swaperów – odpowiedzialnych za kontakt z innymi grupami i resztą sceny. Brak lub ograniczony dostęp do Internetu powodował, że komunikacja pomiędzy grupami odbywała się z wykorzystaniem dyskietek przesyłanych klasycznym listem (tzw. „send”). Twórczość grup obejmowała produkcje demonstrujące możliwości komputera, jak również umiejętności członków grup, tzw. „dema”, ich krótsze wersje, tzw. „intra”, ale również magazyny dyskowe (tzw. „ziny”), będące do dziś cenną skarbnicą wiedzy. Produkcje grup porównywane i oceniane były w ramach konkursów (tzw. „kompotów”, od angielskiego słowa *competition*, w skrócie *compo*) na licznych zlotach ogólnokrajowych oraz międzynarodowych.



Rys. 3.3. Rozklokowanie grup na mapie Polski w okresie rozkwitu sceny – rok 1996

Źródło: <http://atariki.krap.pl/index.php/Scena>

Mimo mocnej już na rynku w połowie lat 90-tych obecności komputerów PC, w dalszym ciągu ośmiobitowe Atari znajdowało ciekawe zastosowania użytkowe, poczynając od niezbyt nowatorskich jak domowa książka teleadresowa, czy baza wydatków, przez mikro-poligrafię na potrzeby gazetek szkolnych, oprogramowanie do obsługi wypożyczalni kaset video, terminal radiowej sieci wymiany danych Packet Radio, sterownik kolejki PIKO, na miernikach laboratoryjnych kończąc. W roku 1999, wydział medyczny Uniwersytetu Karola w Pradze w dalszym ciągu wykorzystywał Atari 800XL wyposażone w magnetofon, ploter oraz sondy gamma, aby w szybki i skuteczny sposób diagnozować wady serca<sup>16</sup>. Do dziś w wielu warsztatach samochodowych wykorzystywane są urządzenia wspomagające wyważanie kół, w których kontrolerem również jest Atari 800XL.

Wyłączając jednakże nieliczne przypadki, których przykłady podano powyżej, obecnie prawdziwe ośmiobitowe komputery Atari używane są praktycznie wyłącznie przez pozostałe na scenie grupy fascynatów. Obok nich powstaje nowa grupa, rzesza użytkowników emulatorów, budowana z dawnych posiadaczy – kupowanych często za bony Pekao – komputerków, którym artykuł w gazecie, bądź opowieść kolegi przypomniawszy pierwsze informatyczne doświadczenia. Mogą dzięki nim w szybki i łatwy sposób, bez konieczności tworzenia pajęczyny kabli na środku salonu, przywołać wspomnienia sprzed lat, a liczne internetowe archiwa pozwalają szybko odszukać ulubione tytuły. Ciekawym zjawiskiem jest fakt, iż komputerami tymi interesuje się również młodzież, która nie miała szansy używać ośmiobitowców w okresie ich świetności - aktywnym od kilku lat polskim koderem z pokaźnym dorobkiem jest m.in. tegoroczny maturzysta.

Motorem napędowym rozwoju platformy ośmiobitowego Atari jest w chwili obecnej środkowa i wschodnia Europa, głównie Niemcy, Czechy, Słowacja i Polska. W tych krajach powstaje najwięcej oprogramowania, tam opracowywane są nowe rozwiązania sprzętowe. Inny charakter mają grupy fascynatów z zachodniej Europy oraz Stanów Zjednoczonych, reprezentowane głównie przez kolekcjonerów sprzętu. Ze względu na łatwiejszy dostęp do wyprzedawanych magazynów likwidowanych laboratoriów Atari, właśnie w rękach mieszkańców tych regionów skupiona jest większość prototypowego sprzętu, jak przenośny komputer Atari 65XEP, prototypy modelu 1450XLD wyposażonego we wbudowaną stację dysków, modem oraz chip syntezy mowy, firmowa stacja dyskietek 3'5" XF351 czy imponujący system kart rozszerzeń Atari 1090.

---

<sup>16</sup> Na podstawie: S.T.A.R., *Atari Cuida Tu Salud*, Matranet, Issue #03 – Febrero 2002

## 3.2. ARCHITEKTURA SPRZĘTOWA

Rozdział ten prezentuje wybrane elementy architektury Atari, których dokładniejsza znajomość została uznana za istotną dla realizowanego projektu.

### 3.2.1. CPU

Wszystkie ośmiobitowe komputery Atari są wyposażone w taki sam 8-bitowy procesor 6502, nie licząc drobnych modyfikacji w przeznaczonym dla serii XL/XE typie 6502C (kryptonim *Sally*) zastępujących dodatkowe układy znane z serii 400/800, zatrzymujące procesor na czas niezbędny koprocesorowi graficznemu ANTIC na dostęp do pamięci. niesprawdzone są pogłoski o tym jakoby istniała na rynku seria komputerów Atari wyposażonych w procesor 65C02 – pomyłka wynika prawdopodobnie z przestawienia występującej w obu nazwach litery C. Procesor posiada 16-bitową magistralę adresową pozwalającą zaadresować liniowo 64 kB pamięci. Trzy rejestry – rejestr ogólnego przeznaczenia (akumulator) oraz dwa rejestry indeksowe X i Y – są wyłącznie 8-bitowe.

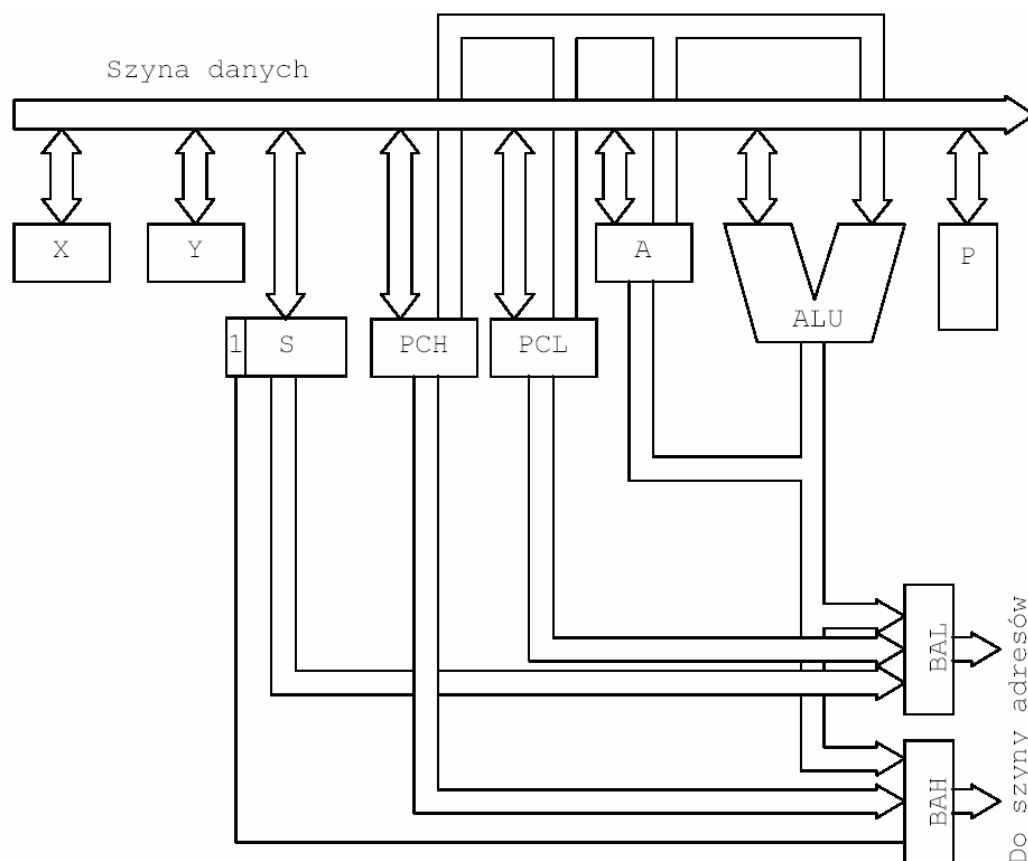
Stos procesora ma pojemność 256 elementów o rozmiarze jednego bajta i jest budowany na pierwszej stronie pamięci, tj. od adresu 0x0100 do 0x01FF. Dziewięciobitowy wskaźnik stosu S wskazuje zawsze pierwsze wolne miejsce w strukturze. Stos wypełniany jest od komórki 0x01FF w dół. Przepelnienie stosu nie skutkuje niczym poza utratą najdawniej złożonych na nim elementów, gdyż wskaźnik stosu po osiągnięciu wartości 0x100 w następnej kolejności osiągnie wartość 0x1FF sprowadzając sytuację do analogicznej, jak przy stosie pustym. Najstarszy, dziewiąty bit rejestru jest na stałe ustawiony.

Procesor posiada 56 udokumentowanych rozkazów, oraz pewną liczbę rozkazów nieudokumentowanych (tzw. rozkazów nielegalnych). Rozkazy nielegalne stanowią prawdopodobnie artefakty optymalizacji układu i pozwalają zaoszczędzić w niektórych przypadkach kilka cykli procesora, jednakże ich używanie nie jest zalecane – wykorzystanie nieudokumentowanych rozkazów powoduje niekompatybilność z procesorem 65816.



W komputerach Atari w systemie PAL procesor 6502 taktowany jest z częstotliwością 1.77MHz (wartość ta dla komputerów w systemie NTSC wynosi 1.79MHz). Ze względu na fakt, iż jeden cykl rozkazowy zużywa zawsze dokładnie jeden cykl zegarowy, mimo niższej częstotliwości taktowania jest on szybszy niż stosowane w tym samym czasie procesory Z80 taktowane z częstotliwością 4MHz, w których jeden cykl rozkazowy zużywa wiele cykli zegarowych. Rozkazy 6502 wymagają 2 do 7 cykli zegarowych, natomiast rozkazy Z80 od 4 do 21 cykli<sup>17</sup>. Pięć z trzynastu dostępnych trybów adresowania 6502 to tryby tzw. strony zerowej, czyli odnoszące się do obszaru w pamięci o adresach 0x0000 do 0x00FF, argument dla rozkazu tego trybu może więc być jednobajtowy (starszy bajt adresu jest domyślnie zerem). Rozkazy tego trybu są zatem krótsze oraz szybsze, dając efektywnie 256 dodatkowych rejestrów pracujących z większą szybkością niż reszta pamięci. O szybkości pracy procesora decyduje również duża ilość szybkich rozkazów jednobajtowych.

Schemat struktury wewnętrznej procesora obrazuje poniższy diagram:



Rys. 3.4. Budowa mikroprocesora 6502

Źródło: J. Ruszczyc, *Asembler 6502*, Warszawa 1987, SOETO

<sup>17</sup> J. Ruszczyc, *Asembler 6502*, Warszawa 1987, Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej

Przedstawione na nim elementy to wspomniane już rejestry (X, Y, A) oraz wskaźnik stosu S, szesnastobitowy licznik rozkazów (z ang. *Program Counter*) z jego młodszym oraz starszym bajtem oznaczonym odpowiednio PCL i PCH, jednostka arytmetyczno-logiczna ALU (z ang. *Arithmetic Logic Unit*) i 16-bitowy bufor szyny adresów (BAL oraz BAH). Szerszego omówienia wymaga rejestr flag procesora, oznaczony na powyższym schemacie literą P. Każdy bit tego rejestru ma odrębne znaczenie, część z nich steruje pracą procesora, część ustawiana jest w wyniku operacji arytmetycznych bądź logicznych i może być wykorzystywana przy sprawdzaniu warunków odgałęzienia warunkowego (ang. *Branch*):

Bit	Symbol	Nazwa	Funkcja
7	N	Negative	Ustawiany jeżeli ostatnio wykonana operacja lub przesłanie dały w wyniku liczbę ujemną (innymi słowy przyjmuje wartość 7 bita wyniku).
6	V	Overflow	Sygnalizuje nadmiar (przepełnienie) dla liczb w kodzie U2.
5		Niewykorzystany. W procesorze 6502 przyjmuje wartości nieokreślone.	
4	B	Break	Sygnalizuje wystąpienie przerwania po napotkaniu rozkazu BRK.
3	D	Decimal	Ustawiony przełącza ALU w tryb dziesiętny (operacje arytmetyczne wykonywane są w kodzie BCD).
2	I	Interrupt disable	Ustawiony blokuje przerwania IRQ.
1	Z	Zero	Informuje, że wynik ostatniej operacji był zerem.
0	C	Carry	Sygnalizuje nadmiar (przeniesienie) dla liczb w kodzie NB.

Tabela 3.1. Znaczenie bitów rejestru flag procesora

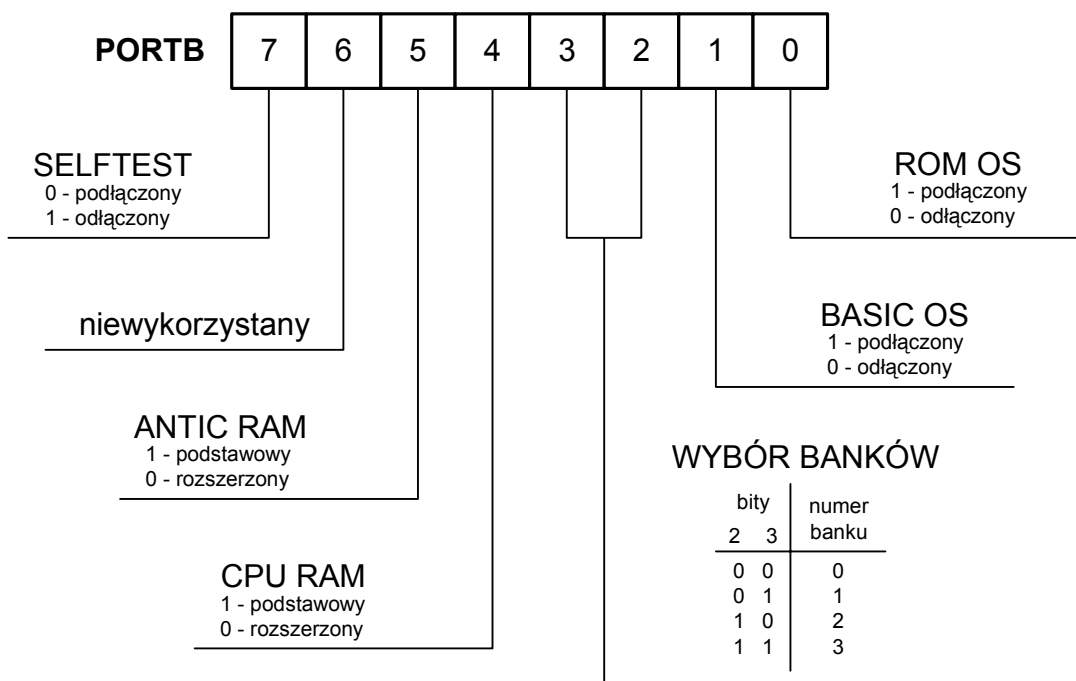
Źródło: opracowanie własne na podstawie: H. Kruszyński, K. Kulpa, *Mikroprocesor 6502 i jego rodzina*, Warszawa 1989, Wydawnictwo Czasopism i Książek Technicznych NOT-SIGMA

### 3.2.2. Pamięć operacyjna

Architektura komputera Atari XL/XE pozwala na liniowe adresowanie 64kB pamięci, taka jest także minimalna wielkość pamięci RAM fizycznie zainstalowanej w komputerach tej rodziny. Ze względu na ograniczenia wynikające z 16-bitowego adresowania, 16kB pamięć ROM zawierająca kod systemu operacyjnego oraz 8kB ROM interpretera języka BASIC muszą również znajdować się w granicach tego obszaru przykrywając część pamięci RAM w momencie, gdy są podłączone.

System obsługi pamięci dodatkowej ponad 64kB został stworzony przez Atari na potrzeby komputera Atari 130XE. Model ten wyposażony został fabrycznie w dodatkowe 64kB pamięci RAM, podłączanej w formie 16kB banków w obszar o adresach 0x4000-0x7FFF. Ten model dodatkowej pamięci przyjęło nazywać się XMS – od Extended Memory Specification.

Zarządzanie pamięcią XMS, jak również podłączanie oraz odłączanie pamięci ROM, odbywa się za pomocą rejestru systemowego PORTB. W komputerach serii 400/800 był to bliźniaczy dla PORTA rejestr układu PIA dla trzeciego oraz czwartego portu joysticka, w serii XE poszczególnym jego bitom przypisano nowe funkcje:



Rys. 3.5. Znaczenie bitów rejestru PORTB w Atari 130XE

Źródło: opracowanie własne, na podstawie I. Chadwick, *130XE Memory Management – How to use the XE's extra 64K*, „Antic” vol. 4 nr 7, listopad 1985, strona 28

Opracowany przez Atari system obsługi pamięci XMS został w późniejszym okresie wykorzystany przez autorów kolejnych rozszerzeń w lekko zmodyfikowanej, lecz zapewniającej kompatybilność, formie – obecnie instalowane rozszerzenia 1MB SIMM również używają bitów rejestru PORTB.

### 3.2.3. Przerwania

Zastosowany w komputerach Atari XL/XE procesor 6502 obsługuje 3 typy przerwań: przerwanie maskowalne (IRQ), przerwanie niemaskowalne (NMI), oraz przerwanie klawisza RESET. Wektory procedur obsługi poszczególnych przerwań (w kolejności: NMI, RESET i IRQ) znajdują się na końcu przestrzeni adresowej procesora, w lokacjach 0xFFFFA-0xFFFFF.

Wywoływanie przerwania RESET następuje przez podanie procesorowi sygnału RST. Obsługa go jest stosunkowo prosta - w momencie wystąpienia wykonywany jest skok do głównej procedury RESET, która z zależności od tego, czy mamy do czynienia z pierwszym wystąpieniem tego przerwania, jakie ma miejsce po włączeniu komputera, czy też kolejnym – jako efekt naciśnięcia klawisza RESET, wykonuje procedury zimnego bądź ciepłego startu<sup>18</sup>.

Źródłem przerwań NMI jest w ośmiobitowym Atari układ ANTIC. Rozróżniamy trzy podtypy przerwań NMI: DLI, VBL oraz NMI RESET. Ostatni z wymienionych podtypów nie jest wykorzystywany przez Atari XL/XE – wywoływany był po naciśnięciu klawisza RESET w komputerach serii 400/800<sup>19</sup>. Przerwanie DLI – inaczej *Display List Interrupt* – generowane jest przez ANTIC po napotkaniu w swoim programie instrukcji z ustawionym bitem 7<sup>20</sup>. Trzecim z tej grupy jest przerwanie VBL – *Vertical BLank interrupt*. Częstotliwość jego występowania jest różna w zależności od regionu, na jaki dany komputer został wyprodukowany – w przypadku komputerów w systemie PAL wynosi ona 49,86 raza na sekundę, w przypadku systemu NTSC 59,92 raza na sekundę. Pojawia się po zakończeniu tworzenia przez ANTIC obrazu, w momencie wygaszania plamki.

Ostatni typ przerwania to przerwanie maskowalne IRQ – *Interrupt ReQuest*. Źródłem przerwania tego typu mogą być układy POKEY oraz PIA, szyna równoległa – przerwanie pochodzące od tzw. nowych urządzeń, lub też sam procesor – po napotkaniu rozkazu BRK. Ustawienie znacznika I w rejestrze znaczników procesora spowoduje zablokowanie przerwań

---

<sup>18</sup> Na podstawie W. Zientara, *Mapa pamięci Atari XL/XE – Podstawowe procedury systemu operacyjnego*, Warszawa 1988, Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej

<sup>19</sup> Na podstawie K. M. Kokoszkiwicz, *NMI*, atariki.krap.pl, 2005, <http://atariki.krap.pl/index.php/NMI>

<sup>20</sup> Na podstawie I. Chadwick, *Mapping The Atari – Revised Edition*, Greensboro 1985, COMPUTE! Publications, Inc.

IRQ. W procesorach 6502 (ale również w ich następach 65C02 oraz 65C816) priorytet przerwania ustalany jest na drodze programowej – procedura obsługi sprawdza w kolejności wszystkie potencjalne źródła jego wystąpienia, począwszy od najwyższego priorytetu:

Przerwanie	Nazwa	Opis
SERIN	Serial Input	Przerwanie odczytu danych z szyny szeregowej
PIRQ	Parallel IRQ	Przerwanie wywołane przez nowe urządzenia
SEROUT	Serial Output	Przerwanie zapisu na szynę szeregową
XMTDONE	Transmit Done	Przerwanie końca transmisji szeregowej
TIMER1	Timer 1	Przerwanie zegara nr 1 układu POKEY
TIMER2	Timer 2	Przerwanie zegara nr 2 układu POKEY
TIMER4	Timer 4	Przerwanie zegara nr 4 układu POKEY
KBDIRQ	Keyboard IRQ	Przerwanie klawiatury
BREAKIRQ	Break key IRQ	Przerwanie klawisza Break
PROCEED	PROCEED	Przerwanie wywołane przez PORT A układu PIA; sygnał ten wyprowadzony jest na gniazdo SIO jako pin <i>PROCEED</i>
INTER	INTERRUPT	Przerwanie wywołane przez PORT B układu PIA; sygnał ten wyprowadzony jest na gniazdo SIO jako pin <i>INTERRUPT</i>
BRKIRQ	Break IRQ	Przerwanie programowe wywołane rozkazem BRK.

Tabela 3.2. Przerwania IRQ w kolejności priorytetów

Źródło: K. M. Kokoszkiewicz, *IRQ*, atariki, 2005, <http://atariki.krap.pl/index.php/IRQ>

Każdy z powyższych rodzajów przerwania może być z osobna włączony bądź wyłączony programowo.

#### 3.2.4. Procesor graficzny ANTIC

ANTIC jest układem w pełni zasługującym na miano procesora graficznego. Posiada własny program (zwany *Display List*), własny zestaw rozkazów oraz bezpośredni dostęp do pamięci komputera. Aby uniknąć konfliktów, na czas swojej pracy zatrzymuje CPU i przejmuje kontrolę nad magistralami systemu. Dostępne rozkazy pozwalają utworzyć obraz składający się z dowolnej kombinacji linii sześciu trybów znakowych oraz ośmiu trybów graficznych. Dla trybów znakowych dodatkowo można wykorzystać kilka sprzętowo wspomaganych efektów, jak odwracanie znaków do góry nogami, odwrócenie kolorów czy tłumienie znaków z ustawionym najstarszym bitem.

Poza generacją obrazu ANTIC odpowiada również za odświeżanie pamięci dynamicznych RAM, oraz jest źródłem przerw nie maskowalnych (NMI). Nie zajmuje się natomiast generacją sygnału wizyjnego – przekazuje jedynie dane do układu GTIA.

### 3.2.5. POKEY

POKEY jest wielofunkcyjnym układem wejścia/wyjścia. Pełni rolę układu UART, zamieniając podawane mu przez system bajty w transmitowane z odpowiednią, programowalną prędkością sygnały elektryczne, a także składając bajty z otrzymywanych na wejściu bitów danych. Prędkość transmisji może być regulowana w zakresie od 13,6 bps do 126674,7 bps, z 65534 wartościami wewnątrz tego zakresu<sup>21</sup>.

Jest również czterokanałowym układem generującym dźwięk - dla każdego z kanałów możliwa jest niezależna regulacja częstotliwości, szumu oraz głośności. Obsługuje analogowe manipulatory – tzw. wioselka (z ang.: *paddle*), klawiaturę, posiada cztery układy zegarowe, a także generator liczb pseudolosowych. Układ POKEY to główne źródło przerw IRQ<sup>22</sup>.

## 3.3. URZĄDZENIA ZEWNĘTRZNE

Urządzenia peryferyjne przeznaczone dla 8-bitowego Atari możemy podłączać wykorzystując jeden z trzech typów przewidzianych przez projektantów komputera interfejsów:

- SIO – magistrala szeregową, umożliwiającą pracę zarówno w trybie asynchronicznym, jak i synchronicznym, z prędkościami do około 115200 bps,
- PBI/ECI – złącze szyny równoległej z wyprowadzonymi sygnałami procesora oraz liniami danych, umożliwia pracę z wysokimi prędkościami, limitowanymi wydajnością procesora. PBI jest standardem złącza wprowadzonym w serii XL, ECI jest uzupełnieniem do funkcjonalnego odpowiednika PBI portu cartridge’a w serii XE,
- porty joysticka – dwa (w serii 400/800 cztery) wielofunkcyjne porty analogowo/cyfrowe.

---

<sup>21</sup> Na podstawie: K. M. Kokoszkiewicz, *Pokey*, atariki.krap.pl, 2005, <http://atariki.krap.pl/index.php/Pokey>

<sup>22</sup> Na podstawie: K. M. Kokoszkiewicz, *Pokey*, atariki.krap.pl, 2005, <http://atariki.krap.pl/index.php/Pokey>

Znakomita większość peryferiów wyprodukowanych przez Atari to urządzenia szeregowo, takie jak stacje dysków, magnetofony, drukarki i wielofunkcyjne interfejsy RS-232C lub Centronics. Poza fabrycznymi produktami, najpopularniejszymi urządzeniami zewnętrznymi obecnie są interfejsy SIO2IDE – umożliwiające podłączanie dysków IDE oraz kart Compact Flash jako stacji dysków dużej pojemności, a także wspomniane w dalszej części rozdziału interfejsy SIO2PC.

Urządzenia podłączane przez porty joysticka to głównie manipulatory: joysticki, wiosełka, tabliczki graficzne, myszy, trackballe czy pióra świetlne. Z tych portów korzystają również urządzenia pomiarowe, przystawki sterujące oraz magnetofonowe systemy turbo. Ciekawostką jest interfejs Atari XEP 80, oferujący możliwość podłączenia drugiego monitora i wyświetlania na nim dodatkowego ekranu tekstowego o szerokości 80 kolumn (tryby tekstowe Atari są maksymalnie 40 kolumnowe). Ze względu na niską prędkość przesyłania danych rozwiązanie to nie doczekało się wielu zastosowań, jednym z bardziej ciekawych jednakże jest uruchamianie debuggera na ekranie dodatkowym, kiedy na ekranie podstawowym oglądamy efekt działania śledzonego programu – funkcjonalność ta dostępna jest w pakiecie *MAE*.

Złącza PBI/ECI są najatrakcyjniejsze pod względem prędkości, dlatego też wykorzystuje je większość dostępnych interfejsów SCSI oraz IDE, również polska konstrukcja stacji dysków 3.5". Jedynym fabrycznym produktem Atari wykorzystującym ten standard jest moduł powiększający pamięć w modelu 600XL. Planowana była produkcja podłączanego za pomocą PBI systemu kart rozszerzeń Atari 1090, z szerokim wyborem kart rozszerzeń, m.in.: rozszerzenia pamięci, programator pamięci EPROM, karta syntezy mowy, szybki interfejs 80-kolumnowy, karta sieciowa Ethernet, emulator Apple, karta CP/M, modem, zegar czasu rzeczywistego, czy kontroler twardego dysku. Niestety, urządzeń wyprodukowano jedynie około 50 sztuk, znane są prototypy trzech typów kart z powyższej listy<sup>23</sup>.

Dla urządzeń PBI projektanci systemu przewidzieli bardzo wygodną koncepcję New Device, będącą praktycznie zaimplementowanym na początku lat 80-tych modelem plug-and-play. Nowe urządzenie podłączane do komputera nie wymaga żadnych dodatkowych sterowników ani konfiguracji aby poprawnie funkcjonować w systemie – w wydzielony obszar pamięci przez system operacyjny automatycznie podłączany jest ROM urządzenia zawierający kod niezbędny do jego obsługi.

---

<sup>23</sup> Na podstawie: <http://atariki.krap.pl/index.php/1090>

Jest również czwarta grupa urządzeń, wykorzystująca do komunikacji z komputerem złącze cartridge'a, przewidziane przez producenta jako gniazdo na moduły z oprogramowaniem. Do pracy potrzebne jest uprzednie wczytanie sterowników dla danego urządzenia, gdyż nie można w tym przypadku wykorzystać mechanizmu New Device znanego z urządzeń PBI. Urządzeniami z tej grupy są m.in. programatory pamięci EPROM, przetworniki analogowo/cyfrowe, zegary czasu rzeczywistego, a nawet niezbyt udana, ale popularna na zachodzie implementacja interfejsu IDE – niezbyt udana dlatego, iż aby z niego skorzystać, należy uprzednio wczytać z dyskietki bądź kasety sterownik do jego obsługi.

### **3.4. ATARI OS**

System operacyjny komputerów serii Atari XL/XE jest kontynuacją systemu dla serii komputerów Atari 400/800. Jest z nim również wstecznie kompatybilny, pod warunkiem że przy tworzeniu oprogramowania autor wykorzystywał jedynie udokumentowane sposoby odwoływania się do procedur systemowych. Napisany został w sposób otwarty, skoki do istotnych elementów systemu odbywają się zawsze poprzez wektory w pamięci RAM, które użytkownik może dowolnie modyfikować, zastępując lub uzupełniając części systemu własnymi procedurami.

Wewnątrz systemu zaszyte są definicje podstawowych urządzeń, jak magnetofon, drukarka oraz edytor, będący połączeniem sterowników ekranu oraz klawiatury. W odróżnieniu od przypadku młodszego konkurenta ośmiobitowego Atari, Commodore 64, nie popełniono błędu i nie umieszczono w systemie złożonych procedur obsługi stacji dysków, zamiast tego wyposażając system w prosty mechanizm automatycznego startu z napędu dysków – umożliwiając doczytanie właściwych procedur już z podłączonego napędu. Dzięki temu użytkownicy komputerów Atari mogą wedle zapotrzebowań wybierać pomiędzy różnymi dyskowymi systemami operacyjnymi (DOS), realizującymi ten sam zestaw funkcji podstawowych, ale różniącymi się złożonością, wielkością w pamięci, formą interfejsu użytkownika, czy dodatkowymi funkcjami.



### 3.4.1. Podsystem wejścia/wyjścia

Część Atari OS zwana podsystemem wejścia/wyjścia CIO (z ang. *Central Input/Output*) jest zestawem procedur oraz struktur danych zapewniających aplikacjom zuniifikowany sposób dostępu do urządzeń zewnętrznych. Dostęp realizowany jest z wykorzystaniem 8 uniwersalnych kanałów, opisanych strukturą IOCB (z ang. *Input/Output Control Block*):

Bajt	Etykieta	Opis
0	ICCHID	Numer identyfikacyjny, będący jednocześnie indeksem wpisu w tablicy handlerów (sterowników) wskazującego dane urządzenie. Wartość \$FF oznacza, że kanał jest zamknięty.
1	ICDNO	Numer urządzenia podanego przez użytkownika (np. 3 dla „D3:NAZWA”), lub 1 gdy nie podano numeru urządzenia (np. „E:”).
2	ICCMD	Kod żądanej operacji (rozkaz); ustawiany przez program. Dozwolone kody: <ul style="list-style-type: none"><li>- \$03 – OPEN – otwarcie pliku</li><li>- \$05 – GET RECORD – odczyt danych w trybie tekstowym</li><li>- \$07 – GET BYTES – odczyt danych w trybie binarnym</li><li>- \$09 – PUT RECORD – zapis danych w trybie tekstowym</li><li>- \$0B – PUT BYTES – zapis danych w trybie binarnym</li><li>- \$0C – CLOSE – zamknięcie pliku</li><li>- \$0D – STATUS – odczyt statusu handlera</li></ul> Wszystkie pozostałe kody o wartościach powyżej 13 (\$0D) są traktowane jako kody operacji specjalnych i są przekazywane bezpośrednio do handlera urządzenia, bez interpretacji przez system.
3	ICSTAT	Status operacji ustawiany automatycznie po jej wykonaniu, czyli wartość 1 gdy sukces, lub ujemny kod błędu.
4-5	ICBUFA	Adres bufora dla operacji, ustawiany przez użytkownika. Dla OPEN, STATUS i operacji specjalnych powinien wskazywać nazwę pliku.
6-7	ICPUTB	Zmniejszony o 1 adres procedury wysyłania 1 bajtu do urządzenia. Ustawiany automatycznie przez system.
8-9	ICBUFL	Wielkość bufora dla danej operacji, ustawiany przez program użytkownika. Jeśli wynosi 0, to bajt przekazywany jest w akumulatorze.
10	ICAX1	Pierwszy bajt pomocniczy. Przy otwieraniu pliku oznacza rodzaj dostępu: <ul style="list-style-type: none"><li>- \$04 – odczyt</li><li>- \$08 – zapis</li><li>- \$09 – dopisywanie</li><li>- \$0C – zapis i odczyt</li></ul>
11	ICAX2	Drugi bajt pomocniczy. Jego znaczenie jest zależne od handlera.

12	ICAX3	Trzeci bajt pomocniczy. Jego znaczenie jest zależne od handlera.
13	ICAX4	Czwarty bajt pomocniczy. Jego znaczenie jest zależne od handlera.
14	ICAX5	Piąty bajt pomocniczy. Jego znaczenie jest zależne od handlera.
15	ICAX6	Szósty bajt pomocniczy. Jego znaczenie jest zależne od handlera.

Tabela 3.3. Struktura IOCB

Źródło: K. M. Kokoszkiewicz, *CIO*, atariki.krap.pl, 2005

[http://atariki.krap.pl/index.php/CIO#Struktura\\_IOCB](http://atariki.krap.pl/index.php/CIO#Struktura_IOCB)

Dostęp odbywa się przez wypełnienie IOCB dla wybranego kanału, załadowanie jego pomnożonego przez 16 numeru do rejestru X oraz wywołanie głównej procedury CIO, JCIOMAIN. Ewentualną daną wejściową należy przed wywołaniem umieścić w akumulatorze. Kod rezultatu operacji zostanie zwrócony w rejestrze Y.

Dane o urządzeniach dostępnych w systemie przechowuje umieszczona na trzeciej stronie pamięci tablica HATABS (*Hardware Address Table*). Ma pojemność 11 trybajtowych pozycji, zawierających literowy kod urządzenia (np.: E dla edytora ekranowego, C dla magnetofonu kasetowego, K dla klawiatury czy D dla stacji dysków) oraz dwubajtowy adres wektora wskazującego tablicę adresową sterownika:

Bajt	Wpis	Funkcja
0	1	Wektor do procedury otwarcia pliku (OPEN-1)
2	2	Wektor do procedury zamknięcia pliku (CLOSE-1)
4	3	Wektor do procedury odczytu danych (GET-1)
6	4	Wektor do procedury zapisu danych (PUT-1)
8	5	Wektor do procedury odczytu statusu (STATUS-1)
10	6	Wektor do procedury operacji specjalnych (SPECIAL-1)
12	Instrukcja skoku JMP do procedury inicjowania handlera	
15	Niewykorzystany	

Tabela 3.4. Struktura tablicy adresowej sterownika

Źródło: W. Zientara, *Mapa pamięci Atari XL/XE – Procedury wejścia/wyjścia*, Warszawa 1988, SOETO

Podczas inicjowania komputera 5 wpisów w tablicy wypełniane jest wartościami z ROMu systemu. Są to wpisy dla urządzeń C, S, E, K i P (nie wspomniane do tej pory S oznacza ekran, a P drukarkę). Pozostałe 6 pozycji jest do dyspozycji programisty. Ze względu na fakt, iż tablica HATABS umieszczona jest w całości w pamięci RAM, możliwa jest również modyfikacja urządzeń instalowanych w niej przez procedury Atari OS. Tablica

przeszukiwana jest od ostatniej pozycji, jeżeli więc w tablicy znajdują się dwa wpisy o takiej samej nazwie urządzenia, pierwszy z nich będzie przez procedury systemu ignorowany.

Podsystem posiada dość istotną wadę – projektanci założyli, iż na wejściu do procedury głównej CIO, nawet w celu otwarcia dostępu do urządzenia, programista musi podać numer kanału. Powoduje to problemy przy wywoływaniu programu z wewnątrz innego, kiedy oba zostały napisane tak, aby wykorzystywać ten sam kanał: jeżeli uruchamiany z wewnątrz program zakłada, że kanał jest wolny, napotka błąd, gdyż kanał został już otwarty przez program wywołujący; jeżeli profilaktycznie zamknie kanał przed próbą otwarcia, będzie funkcjonował poprawnie, ale program nadrzędny po zwróceniu kontroli napotka błąd przy próbie odczytu lub zapisu do zamkniętego przez program „wewnętrzny” kanału.

Sposobem obejścia tej niedogodności jest używanie w pisanych programach procedury wykrywającej numer wolnego kanału zanim nastąpi próba jego otwarcia. Poniżej przykładowa procedura realizująca ten cel, autorstwa K. M. Kokoszkiewicza<sup>24</sup>:

```
lookup    ldx #$00
           ldy #$01
loop      lda icchid,x
           cmp #$ff
           beq found
           txa
           clc
           adc #$10
           tax
           bpl loop
           ldy #-95 ;kod błędu "TOO MANY CHANNELS OPEN"
found     rts
```

Jeżeli procedura zakańcza swoje funkcjonowanie z ustawionym znacznikiem N, oznacza to, iż znaleziono wolny kanał i jego numer pomnożony przez 16 przekazywany jest w rejestrze X.

#### 3.4.2. Procedury odczytu wstępnego

Po włączeniu komputera lub naciśnięciu klawisza RESET wywoływana jest procedura

---

<sup>24</sup> K. M. Kokoszkiewicz, *Programowanie: Jak wyszukać pierwszy wolny IOCB*, atariki.krap.pl, 2005

o tej samej nazwie. W zależności czy jest to pierwsze uruchomienie (tzw. zimny start), czy start ciepły wywołany wciśnięciem wspomnianego klawisza, wykonywana jest pełna inicjalizacja systemu wraz z zerowaniem pamięci podstawowej i weryfikacją sumy kontrolnej ROM, lub tylko odtworzenie istotnych obszarów pamięci, jak wektory przerwań, tabela sterowników, a następnie otwarcie edytora ekranowego.

Częścią procedury wykonywanej podczas zimnego startu jest próba wykonania odczytu wstępnego (ang. *boot*) z kasety (jeżeli podczas uruchamiania systemu wciśnięty był klawisz START konsoli) i ze stacji dysków elastycznych. Obie procedury są bardzo zbliżone, różnią się tylko wywołaniami procedur specyficznych dla medium, z którego pobierane są dane. Dalszy opis będzie dotyczył procedury odczytu wstępnego z dysku, procedura kasetowa różni się tylko sposobem obsługi błędów – w przypadku dysku wykonywana jest nieskończona ilość prób, w przypadku kasety jest to niemożliwe i odczyt zakańczany jest niepowodzeniem. Na początku procedury wczytywany jest cały pierwszy sektor dysku w stacji numer 1 (trzy pierwsze sektory, tzw. boot sektory, są zawsze, niezależnie od gęstości w jakiej sformatowana jest dyskietka, sektorami 128 bajtowymi). Sześć pierwszych bajtów stanowi nagłówek o następującej strukturze:

Bajt	Etykieta	Opis
0	DFLAG	Pole flag. Niewykorzystywane przez system, zapisywane do komórki DFLAGS o adresie \$240.
1	DSECCNT	Długość kodu rozruchu (łącznie z nagłówkiem), podana w 128 bajtowych sektorach. Zapisywana do komórki DBSECT (\$241).
2-3	BOOTAD	Adres miejsca w pamięci do którego zostanie wczytany kod rozruchu (łącznie z nagłówkiem). Zwyczajowo dla DOS jest to adres \$700. Wartość jest zapisywana do komórki BOOTAD (\$242-\$243) oraz tymczasowo do RAMLO (\$4-\$5).
4-5	DOSINI	Adres początku procedury rozruchu pod który należy skoczyć po wczytaniu całości kodu. Zapisywana w wektorze DOSINI (\$C-\$D). W przypadku gdy rozruch nastąpił z kasety, zawartość wektora DOSINI jest również przepisywana do wektora CASINI (\$2-\$3).

Tabela 3.5. Struktura nagłówka rozruchu (*boot*)

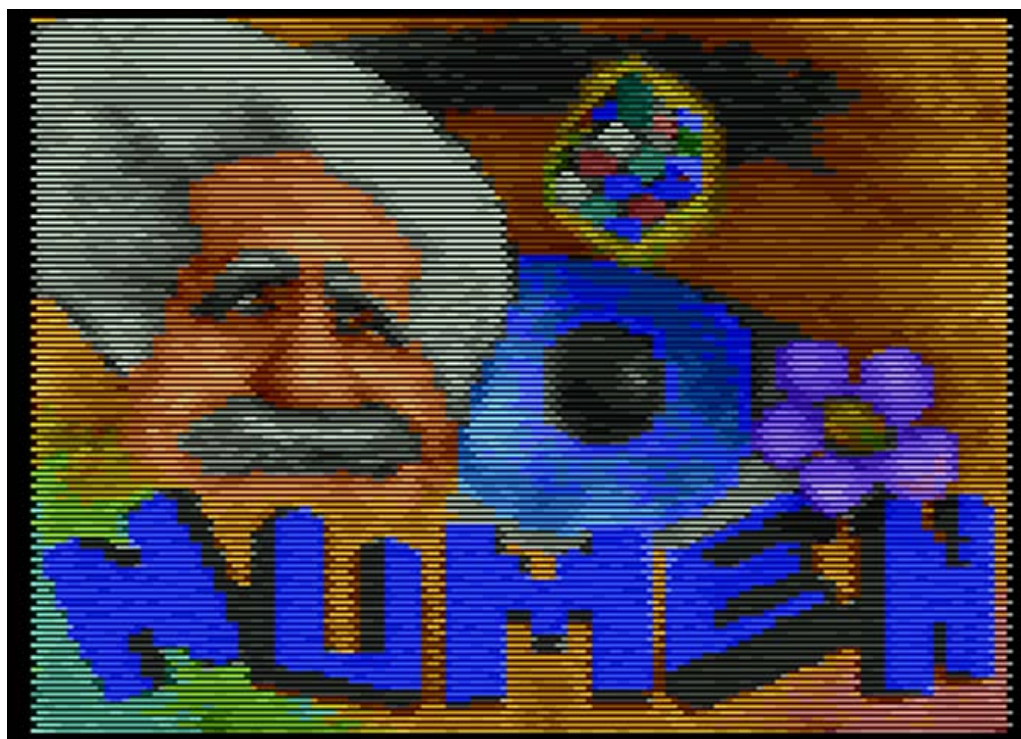
Źródło: opracowanie na podstawie W. Zientara, *Mapa pamięci Atari XL/XE – Podstawowe procedury systemu operacyjnego*, Warszawa 1988, SOETO oraz I. Chadwick, *Mapping The Atari – Revised Edition*, Greensboro 1985, COMPUTE! Publications

Wektor DOSINI wykorzystuje również procedura ciepłego startu – w najczęstszym przypadku, kiedy znajduje się tam adres skoku do procedury dyskowego systemu operacyjnego, dba ona o umieszczenie odpowiednich wpisów dotyczących napędu dyskietek w tablicy sterowników.

### 3.5. ROZWÓJ PLATFORMY

#### 3.5.1. Rozwój oprogramowania

Na ośmiobitowe Atari w dalszym ciągu powstaje nowe oprogramowanie. Nowe demo pisane są przeważnie na – co prawda rzadsze, ale regularne – złoty. Największym tego typu wydarzeniem jest odbywający się corocznie od 1993 roku letni zlot QuaST w miejscowości Orneta. Jedno z najlepszych dem na ośmiobitowe Atari, Numen grupy Taquart, zostało po raz pierwszy wystawione w roku 2002, zawiera między innymi gotowy engine do nigdy nie dokończonej gry w stylu znanego z komputerów PC Doom'a<sup>25</sup>.



Rys. 3.6. *Numen* autorstwa Taquart – ekran tytułowy

Źródło: opracowanie własne

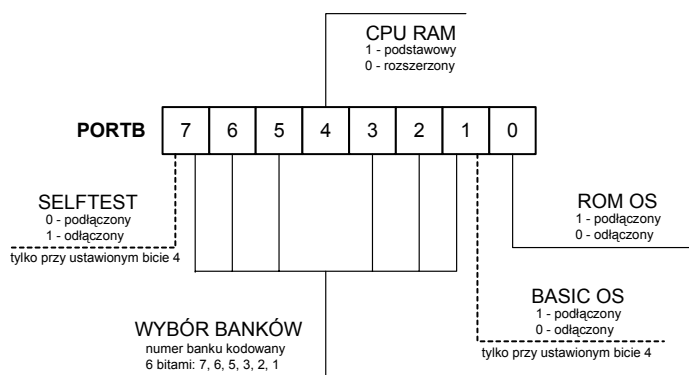
---

<sup>25</sup> Na podstawie: <http://numen.scene.pl/>

W dalszym ciągu powstają nowe gry. Czasem są to oryginalne pomysły autorów, czasem implementacje dawnych hitów z innych platform – jak choćby zwycięzca konkursu ABBUC Programming Contest 2004, gra Dyna Blaster – implementacja znanej z Amigi i PC gry zręcznościowej. Obecne gry wykorzystują wszystkie poznane przez lata produkcji scenowych triki, są więc przeważnie barwniejsze oraz dynamiczniejsze niż komercyjne produkcje z przełomu lat 80-tych i 90-tych. Wykorzystywane są również nowinki sprzętowe – gry odgrywają dźwięk stereo, obsługują 8 joysticków za pomocą interfejsu *MultiJoy*, bądź też pozwalają na grę w sieci spiętych ze sobą za pomocą *Game Link II* komputerów.

### 3.5.2. Modyfikacje sprzętowe

Najczęściej wykonywaną modyfikacją komputerów Atari serii XL/XE jest rozszerzenie pamięci operacyjnej. Dawniej motywacją było ułatwienie procesu kopiowania dyskietek – przy rozszerzeniu RAM do 192kB kopierzy były w stanie załadować do bufora w całości jedną stronę dyskietki nagranej w podwójnej gęstości. Na początku XXI w. standardem stało się 320kB wymagane przez większość produkcji demosceny, obecnie natomiast w nowowyposażanych komputerach Atari dominuje rozszerzenie do 1MB RAM, oparte na tanich kościach SIMM. Zapewnia ono dostęp do 64 dodatkowych banków pamięci. Rozwiązanie jest zgodne z modelem zaproponowanym w 130XE, z kilkoma wyjątkami: niemożliwy jest dostęp do dodatkowej pamięci z poziomu wbudowanego interpretera BASICa ani SELF TESTu, niemożliwy jest również wybór pomiędzy pamięcią podstawową a rozszerzoną osobno dla CPU i ANTICa. Funkcje poszczególnych bitów w rejestrze PORTB zostały zmienione w sposób następujący:



Rys. 3.7. Znaczenie bitów rejestru PORTB w *1MB Pasiu SIMM Expansion*

Źródło: opracowanie własne

Kolejną popularną modyfikacją jest dodanie drugiego układu POKEY. Przeróbka, której autorem jest Ch. Steinman, dodaje dodatkowe 4 kanały dźwiękowe, efektywnie umożliwiając generowanie dźwięku stereo. Pozostałe funkcje dodatkowego układu nie są wykorzystywane.

Stosunkowo nowym rozszerzeniem o dużym potencjale jest opracowany przez M. Pasiecznika *WARP4*. Modyfikacja polega na wymianie istniejącego procesora 6502C na jego produkowanego przez Western Design Center 16-bitowego następcę, 65C816. Procesor ten posiada dwa tryby pracy, tryb emulacji, w którym zachowuje pełną zgodność z 6502 (z wyłączeniem nielegalnych rozkazów) oraz tryb naturalny, w którym rejestry mogą mieć wielkość 16 bitów, strona zerowa może być przesuwana wektorem po całym obszarze pierwszych 64kB pamięci RAM, stos może mieć wielkość do 64kB, a przestrzeń adresowa zwiększa się do 16MB. Miejsce nielegalnych rozkazów 6502 zajął zestaw rozszerzający istniejącą pulę o rozkazy nowych trybów adresowania oraz kilka rozkazów przydatnych w obu trybach pracy komputera (jak na przykład transfer bezpośredni pomiędzy rejestrami indeksowymi, bądź też bezwarunkowy rozkaz *BRANCH*). Aby w pełni wykorzystać możliwości nowego procesora, płyta z rozszerzeniem zawiera dodatkowe 1MB pamięci RAM, podłączone w trybie liniowym, zaraz za podstawowymi 64kB pamięci, dostępne w trybie naturalnym 65C816. Korzystając z faktu, iż 65C816 może być taktowany zegarem nawet do 16MHz, dostęp do tej pamięci odbywa się z podwyższoną prędkością – w prototypie było to 4MHz (stąd nazwa), w komputerze autora pracuje rozwiązanie taktowane zegarem 7MHz. Ze względu na częstotliwość pracy procesora ANTIC, dostęp do podstawowego zakresu RAM musi odbywać się ze zwykłą prędkością. Korzystać z przyspieszenia mogą więc tylko aplikacje wykorzystujące liniowy RAM powyżej 0x00FFFF. Niedogodności tej pozbawiony jest nowy projekt autora *WARP4*, chwilowo pod roboczą nazwą *Project F7* – nazwa pochodzi od klawisza, który w emulatorze Atari800WinPLus powoduje zwiększenie prędkości emulacji komputera do wartości ograniczonych jedynie wydajnością PC, na którym emulator jest uruchamiany. W opracowanym prototypie obszar początkowych 64kB RAM przykryty jest dodatkową, szybką pamięcią będącą cieniem pamięci oryginalnej<sup>26</sup>. W ten sposób zapis do pamięci odbywa się z normalną prędkością (zapisywane są pamięć oraz jej cień), odczyt wykonywany przez ANTIC odbywa się z normalną prędkością (czytana jest pamięć oryginalna), natomiast odczyt przez CPU odbywa

---

<sup>26</sup> Na podstawie: *Czy to wykonalne?*, atari.area, 2004

się z maksymalną prędkością. Ze względu na to, iż większość operacji procesora to odczyt, efektywne przyspieszenie dotyczy wszystkich aplikacji. Dodatkową różnicą jest tryb pełnej kompatybilności – *Project F7* zakłada zachowanie starego procesora 6502C i używanie go dla programów które zostały napisane z wykorzystaniem rozkazów nieudokumentowanych<sup>27</sup>.

Jednym z prostszych rozszerzeń, a w zasadzie interfejsów, często dla wygody montowanych wewnątrz komputera, jest interfejs *SIO2PC*. Autorem oryginalnej wersji interfejsu oraz jego oprogramowania jest N. Kennedy<sup>28</sup>. Obecna – uproszczona – konstrukcja składająca się z układu MAX232, kilku kondensatorów, rezystora i diody umożliwia podłączenie komputera PC wyposażonego w interfejs RS232C do złącza SIO Atari. Oprogramowanie na PC wykonuje emulację urządzeń podłączanych z wykorzystaniem tego właśnie złącza, w szczególności stacji dysków. Rozwiązanie jest bardzo rozpowszechnione, ze względu na prostotę konstrukcji i niewielki koszt – kształtujący się z okolicach 10 zł wraz z kosztami niezbędnych wtyczek i kabli, a także ze względu na uzyskiwany dzięki niemu uproszczony dostęp do oprogramowania dostępnego w sieci w formacie obrazów dyskietek (ATR) oraz łatwą i szybką możliwość transferu plików pomiędzy obiema platformami

Kolejnym interfejsem o niezwyklej prostocie wykonania jest *Game Link II*, autorstwa grupy AGDA. W swojej najprostszej wersji, pozwalającej na połączenie ze sobą dwóch komputerów Atari, do jego konstrukcji potrzebne są dwa kable SIO (dołączane w zestawie m.in. ze stacją dysków) oraz trzy spinacze biurowe (celem wykonania odpowiednich połączeń sygnałów Serial Input Data, Serial Output Data oraz GND)<sup>29</sup>. Zbudowana w ten sposób sieć może liczyć do 8 komputerów, komunikujących się na zasadzie klient-serwer<sup>30</sup>.

Najnowszym, bo zakończonym i opublikowanym w marcu 2006 roku, jest projekt *Video Board XE* autorstwa electrona z polskiej grupy taquart. Jest to pierwsza na świecie, i co ważniejsze – udana, próba rozszerzenia możliwości graficznych ośmiobitowego komputera Atari. *VBXE* stanowi rozwinięcie układu GTIA, jest kompatybilne w dół, aczkolwiek oryginalne GTIA również jest zachowywane dla zgodności z oprogramowaniem wykorzystującym starą grafikę gracza-pocisku (ang. *player-missile graphics*). Posiada własne 512kB pamięci obrazu, jest w stanie generować obiekty o rozmiarach do 256x256 pikseli w

---

<sup>27</sup> Na podstawie: <http://www.pasiu.krap.pl/index.htm>

<sup>28</sup> Na podstawie: <http://atariki.krap.pl/index.php/SIO2PC>

<sup>29</sup> Na podstawie: The AGDA Group, *GameLink-II Specification*, DataQue Software, 1993

<sup>30</sup> Na podstawie: J. Bernašek, *The Inside of Network Games*, BEWESOFT



256 kolorach z 65536 kolorowej palety w trybie 320x192, ze sprzętowo zrealizowanym przesuwaniem oraz detekcją kolizji. Ilość obiektów nakładanych na obraz zależy od ich rozmiaru, deklarowana wydajność to co najmniej 30 obiektów o rozmiarach 32x32 w 256 kolorach na ramkę (czas pomiędzy przerwaniem VBL).

Pozycje obiektów mogą być zdefiniowane co do jednego piksela w trybie wysokiej rozdzielczości ANTIC. Praca układu nie zabiera czasu procesora, konieczny jest tylko transfer danych dla obiektów do pamięci karty, manipulacja samymi obiektami wymaga następnie tylko wysyłania prostych komend sterujących<sup>31</sup>.



Rys. 3.8. Demonstracja możliwości prototypu karty *Video Board XE*

Źródło: dely, *Video Board XE - projekt w 90% zakończony*, atari.area, 04.04.2006

Dodatkowym atutem jest to, że po uruchomieniu sama karta jest niezaprogramowana, z poziomu systemu operacyjnego Atari należy załadować do niej odpowiedni plik konfiguracyjny – można więc w łatwy sposób uaktualniać oprogramowanie rozszerzenia lub wręcz, zależnie od potrzeb, wykorzystać dodatkową pamięć i moc obliczeniową jako koprocesor do zupełnie innych celów niż akcelerator graficzny<sup>32</sup>.

---

<sup>31</sup> Na podstawie: electron/taquart, *Video Board XE specification*, 2006

<sup>32</sup> Na podstawie: electron, *Video Board XE - projekt w 90% zakończony*, atari.area, 05.03.2006

## 4. ŚRODOWISKO PRACY

Całość pracy nad systemem odbywała się z wykorzystaniem wyłącznie oprogramowania uruchamianego na Atari - przenośny komputer PC służył głównie jako emulator urządzeń Atari, bądź emulator samego komputera. W tym ostatnim przypadku możliwość wykorzystywania przenośnego komputera zastępującego zestaw złożony z jednostki centralnej Atari XEGS, zasilacza, monitora, klawiatury oraz stacji dysków umożliwiała pracę również poza domowym laboratorium.

Przy pracy nie były wykorzystywane krosasemblery ani kroskompilatory języków wyższego rzędu. Mimo dobrze wyposażonego komputera, na którym powstawał kod, został on napisany tak, aby pracował również na nierozszerzonym Atari 65XE.

### 4.1. OPROGRAMOWANIE NA ATARI

Podstawowym narzędziem wykorzystywanym do edycji kodu źródłowego oraz jego asemblacji do formy binarnej był *MAE 1.3*, autorstwa J. Harrisa. Ten potężny pakiet, rozwijany przez autora jeszcze w roku 1999, oferuje użytkownikowi do wyboru 40-, 64- lub 80-kolumnowy edytor z wieloma przydatnymi funkcjami typu: skok do podprogramu z zapamiętaniem miejsca w źródle z którego skok nastąpił, skoki do etykiet czy też makra. Debugger umożliwia śledzenie programu z wykorzystaniem drugiego monitora podłączonego przez interfejs XEP80 i port joysticka, a doskonały makro-asmembler generuje kod również dla procesora 65816, oferuje też asemblację warunkową.

Przydatnymi przy tworzeniu prostych programów narzędziowych oraz do wykonywania niektórych testów okazały się interpretery *Turbo Basica XL 1.5*, autorstwa F. Ostrowskiego, oraz *Multibasics*, autorstwa K. M. Kokoszkiewicza. Komputer Atari oraz emulator pracowały podczas powstawania XEUXa pod kontrolą dyskowego systemu operacyjnego *SpartaDOS X 4.3*, wersja rozwijana obecnie przez grupę DLT.

Inne drobne programy używane podczas pracy to: *Font Designer 5.0* autorstwa LASER soft, *ED 2.9* autorstwa J. B. Wiśniewskiego z późniejszymi poprawkami Truba, *TAR*, *BLOAD*, *UNIFY* oraz *CP* autorstwa K. M. Kokoszkiewicza.

## 4.2. OPROGRAMOWANIE NA PC

Pierwszą pozycję na liście wykorzystywanego oprogramowania na PC zajmuje emulator *Atari800WinPLus* w wersji 4.0, rozwijanej na bazie multiplatformowego *Atari800 D.Firtha*<sup>33</sup>. Kolejnym istotnym fragmentem środowiska pracy jest *The Atari Peripheral Emulator*, napisany przez S. J. Tuckera, emulujący do ośmiu napędów dyskietek plus interfejs szeregowy RS-232C w standardzie Atari 850. Drobnym, ale istotnym programem użytkowym manipulującym obrazami dyskietek formatu ATR jest *MakeATR* autorstwa Soudi.

## 4.3. SPRZĘT

Głównym komputerem użytkowanym podczas pracy nad systemem był Atari XEGS wersja PAL, wyposażony w procesor 65816 – rozszerzenie *WARP4* autorstwa M. Pasiecznika – taktowany zegarem 7MHz przy adresowaniu pamięci liniowej powyżej 0x00FFFF oraz 1.77MHz przy adresowaniu podstawowych 64kB RAM, 1MB RAM XMS + 1MB pamięci liniowej dostępnej w trybie naturalnym 65816, kontroler *IDE JŻ/KMK* według projektu J. Żuka oraz K. M. Kokoszkiewicza, zegar czasu rzeczywistego *ARClock* oraz umożliwiający aktualizację wersji systemu operacyjnego zapisanego z poziomu Atari system *ROM Changer*, oba autorstwa M. Pasiecznika.

Dodatkowo wykorzystywano komputery Atari 65XE oraz Atari 130XE – w celu testowania powstającego kodu na niezmodyfikowanych komputerach oraz jako terminale sieciowe połączone interfejsem GameLink II.

Urządzenia peryferyjne użytkowane to: kabel *SIO2PC* przydatny przy oprogramowywaniu komunikacji ze stacją dysków oraz stacją dysków Atari XF551.

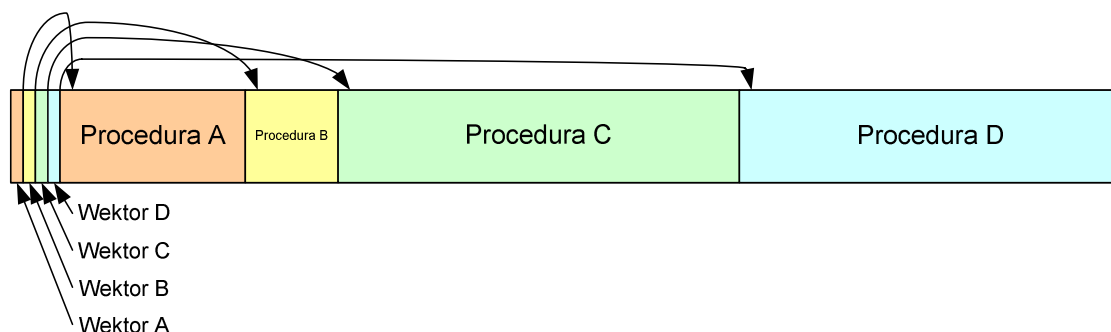
## 4.4. ORGANIZACJA PLIKÓW ŹRÓDŁOWYCH

Kod źródłowy systemu, nie licząc kodu narzędzi, ma objętość ponad 120kB. Ze względu na wygodę pracy – wczytanie 120kB do edytora assemblera nie jest możliwe – a również mając na uwadze przejrzystość kodu systemu, pliki źródłowe podzielone zostały na kilka sekcji, łącząc funkcje systemu o zbliżonym zastosowaniu.

---

<sup>33</sup> Oryginalny emulator był rozszerzany i poprawiany przez wiele grup programistów, obecna wersja rozwijana przez M. Lewandowskiego. Strona domowa projektu: [http://atariarea.krap.pl/PLus/index\\_pl.htm](http://atariarea.krap.pl/PLus/index_pl.htm)

Każdy z modułów zawiera na początku wektory skoków do właściwych procedur znajdujących się w danym module. Bez względu na późniejsze modyfikacje i rozbudowy kodu, wektory zawsze znajdują się w tym samym miejscu pamięci. Dzięki temu modyfikacje kodu modułów nie powodują konieczności przebudowywania wykorzystujących je programów, w tym procesów użytkownika.



Rys. 4.1. Organizacja procedur w ramach modułów

Źródło: opracowanie własne

Adresy wektorów skoków umieszczane są w pliku nagłówkowym – ROBP.MAE. Ten plik, wraz z plikiem ROBI.MAE, zawierającym adresy wskazujące przestrzeń danych, są dołączane za pomocą dyrektywy *include* do pozostałych kodów źródłowych.

## 4.5. TWORZENIE PLIKÓW WYNIKOWYCH

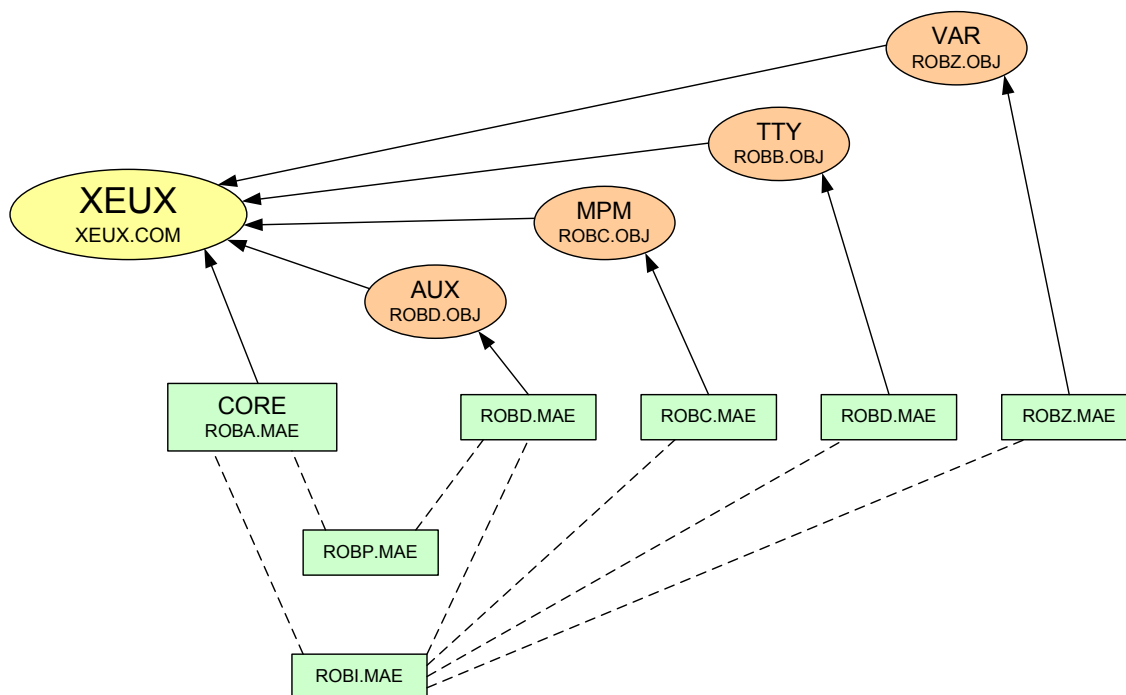
### 4.5.1. Pliki systemu operacyjnego

System operacyjny, jak wspomniano, składa się z czterech modułów dołączanych do kodu wynikowego jądra. Są to:

- **moduł B** (TTY), zawierający procedury obsługi systemu wirtualnych konsol i edytora ekranowego,
- **moduł C** (MPM), zawierający procedury zarządzania pamięcią i kod planisty długoterminowego,
- **moduł D** (AUX), zawierający procedury wspomagające, zawierające elementy obsługi sygnałów, obsługę operacji dyskowych i inicjowania systemu,
- **moduł Z** (VAR), zawierający przestrzeń danych systemowych, wraz z predefiniowanymi stałymi i zmiennymi.

Wszystkie moduły, z wyjątkiem D, są niezależne. Moduł D odwołuje się do procedur alokacji pamięci znajdujących się w module C, jednakże wszystkie te odwołania wykonywane są z użyciem właściwych wektorów, toteż zmiany w C nie powodują konieczności zmian bądź przebudowywania kodu wynikowego modułu D.

Główny kod jądra systemu, łącznie z programem planisty krótkoterminowego, znajduje się w **module A (CORE)**. Do niego w trakcie tworzenia kodu wynikowego dołączane są pozostałe pliki wynikowe modułów, dlatego czynność ta powinna być ostatnią. W efekcie powstaje plik wykonywalny w formacie Atari DOS.



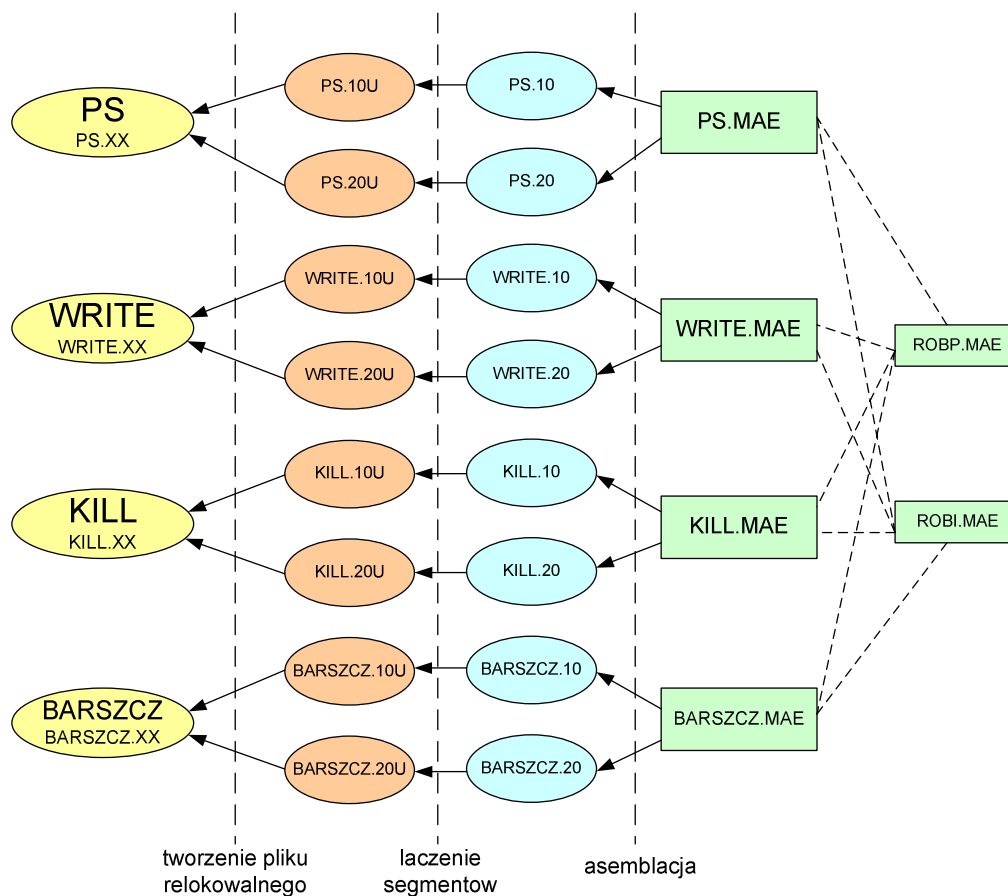
Rys. 4.2. Elementy składowe pliku wynikowego systemu

Źródło: opracowanie własne

W przyszłości, w momencie pełnego wyeliminowania Atari ROM, kod systemu będzie mógł być umieszczony w miejscu starego systemu operacyjnego Atari, po zaprogramowaniu odpowiednio pamięci EPROM, lub przygotowany w formie cartridge'a, wkładanego bezinwazyjnie w przewidziane gniazdo rozszerzeń. Drugi przypadek odłączał będzie pamięć ROM komputera i instalował kod systemu w pamięci RAM znajdującej się w tej samej przestrzeni adresowej.

#### 4.5.2. Pliki procesów

Proces przygotowywania plików wykonywalnych procesów jest szczegółowo opisany w załączniku (Rozdział 8, punkt 2.4). Składa się on z kilku etapów, zilustrowanych na poniższym schemacie:



Rys. 4.3. Proces tworzenia plików wykonywalnych dla systemu XEUX

Źródło: opracowanie własne

Pliki źródłowe wykorzystują wspólne pliki nagłówkowe jako bazę adresów funkcji systemowych i rejestrów dostępnych dla procesów użytkownika. Dzięki mechanizmowi wektorów, raz przygotowane pliki binarne nie wymagają przebudowywania po każdej rozbudowie systemu operacyjnego.

## 5. PODSTAWOWE ELEMENTY SYSTEMU OPERACYJNEGO

### 5.1. PROCES

Jedną z definicji procesu definiuje go jako „moduł działającego programu, charakteryzowany przez pojedynczy sekwencyjny wątek wykonywanego kodu, aktualny stan i związany z nim zbiór zasobów systemowych”<sup>34</sup>. Do zasobów procesu w systemie XEUX zaliczamy zarówno miejsce w pamięci, w którym ulokowany jest kod i dane procesu, ale również stan rejestrów i flag procesora, licznika rozkazów, stosu systemowego oraz strony zerowej. Lokalna jest też informacja dotycząca otwartych plików i urządzeń przechowywana w uniwersalnych kanałach wejścia-wyjścia. Aktualny stan procesu, wraz z informacjami dotyczącymi posiadanych zasobów, przechowywany jest w strukturze danych zwanych stroną bazową procesu, opisaną szczegółowo w dalszej części rozdziału.

Ważnym jest rozróżnienie programu od procesu – jeden program może być uruchomiony jednocześnie w kilku kopiach, ale mimo takiego samego kodu i zapotrzebowania na przestrzeń danych będą one zawsze traktowane jako odmienne procesy.

#### 5.1.1. Strona bazowa procesu

Strona bazowa procesu to struktura danych zawierająca wszystkie kluczowe informacje dotyczące danego procesu. Zawiera następujące informacje:

Pole	Rozmiar	Opis
PR.PID_	1 bajt	Numer identyfikacyjny procesu, z zakresu od 0 do 255. Po osiągnięciu maksymalnej wartości przydział kolejnych numerów rozpoczyna się od 0.
PR.STATE_	1 bajt	Stan procesu. Opisany szczegółowo w dalszej części tego rozdziału.
PR.PRIOR_	1 bajt	Wartość na jaką ustawiany jest kredyt procesu w momencie wznowiania jego pracy przez planistę (zmiana stanu na \$04 – „running”).
PR.CREDI_	1 bajt	Aktualny kredyt procesu – ilość wywołań planisty zanim proces zostanie wyłączone.
PR.STACK_	1 bajt	Wskaźnik stosu procesu – pole wypełniane przy zachowywaniu kontekstu
PR.BANK_	1 bajt	Numer banku pamięci XMS (w tabeli BANKTBL_) w którym znajduje się

---

<sup>34</sup> W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic

		pamięć przydzielona procesowi.
PR.TTY_	1 bajt	Numer konsoli wirtualnej przypisanej do procesu.
PR.PAGE_	1 bajt	Strona pamięci na której rozpoczyna się kod procesu.
PR.ZERO_	1 bajt	Strona pamięci przechowująca stronę zerową procesu.
PR.PARENT_	1 bajt	Numer procesu-rodzica (przechowywanie nie jako PID, ale jako numer pozycji na stronie bazowej). Ustawiane przez planistę długoterminowego (PRLOAD_)
PR.SIGNAL_	1 bajt	Numer sygnału jaki oczekuje na obsługę przez proces. Wypełniane w momencie pojawienia się sygnału dla procesu, wraz z ustawieniem odpowiedniego bitu w statusie procesu.
PR.IOQUE_	1 bajt	Numer urządzenia jakie oczekuje na obsługę przez proces. Wypełniane w momencie pojawienia się żądania obsługi, wraz z ustawieniem odpowiedniego bitu w statusie procesu.

Tabela 5.1. Struktura strony bazowej procesu

Źródło: opracowanie własne

Nazwa procesu, uzyskiwana z nazwy pliku wczytywanego programu, umieszczana jest w dodatkowej tablicy PRNAME\_.

#### 5.1.2. Stos procesora, strona zerowa i przestrzeń IOCB.

Ze względu na brak wektora stosu w procesorze 6502, częścią procesu przełączania kontekstu jest też proces przepisywania stosu systemowego tak, aby każdy z procesów posiadał niezależną jego kopię. Pierwsze procedury zaimplementowane w prototypie systemu kopiowały całość przestrzeni pamięci przeznaczonych na stos – 256 elementów – niezależnie od jego wykorzystania. Dodatkowo osobno do specjalnych struktur odkładane były rejestry, licznik rozkazów oraz flagi procesora.

W aktualnej wersji systemu zachowywany jest tylko fragment pamięci stosu zawierający elementy znajdujące się na nim aktualnie, czyli elementy od wskaźnika stosu do końca pierwszej strony pamięci. Ten sposób ma pewną wadę związaną ze sposobem funkcjonowania stosu, opisanym szerzej w Rozdziale 3 – w przypadku przepełnienia stosu może on nie zostać poprawnie odtworzony i w efekcie spowodować przy przełączaniu procesu skok w dowolne miejsce pamięci. Niestety, nie ma dobrego rozwiązania które zabezpieczyłoby przed niestabilnością w przypadku przepełnienia stosu. Całe szczęście 256 bajtów stosu dla każdego z procesów jest wartością wystarczającą w znakomitej większości przypadków.



Drugim obszarem pamięci, który bezwzględnie powinien być lokalny dla każdego procesu jest strona zerowa, czyli obszar o adresach od 0x0000 do 0x00FF. Umożliwia on wykorzystywanie krótszych i szybszych rozkazów strony zerowej oraz trybów adresowania pośredniego. Jest on odkopiuwany prawie w całości dla każdego z procesów – nietknięte pozostają fragmenty wykorzystywane wyłącznie przez kod jądra.

Ostatnią stroną struktur danych każdego procesu jest przestrzeń zawierająca informacje dotyczące otwartych kanałów komunikacyjnych z urządzeniami zewnętrznymi dla podsystemu XXCIO. Kanałów tych jest 8 i noszą nazwę IOCB (z ang.: *Input Output Control Block*). Ten obszar pamięci pojawił się dopiero w późniejszych wersjach prototypu, wraz z rozwojem rozszerzonego podsystemu obsługi urządzeń.

## 5.2. ZARZĄDZANIE PAMIĘCIĄ

### 5.2.1. Obsługa dodatkowej pamięci komputera

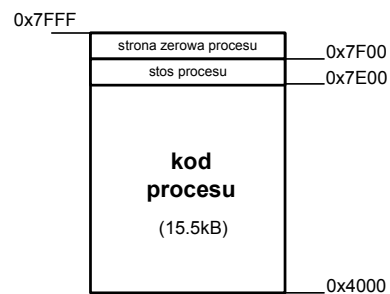
System XEUX dla komputerów wyposażonych w procesor 6502 wspiera obsługę dodatkowej pamięci XMS – dostępnej jako 16kB bloki, podłączane w obszarze 0x4000-0x7FFF. Ze względu na istnienie wielu rozwiązań rozszerzeń, różniących się zarówno wielkością dodatkowej pamięci, jak i przyporządkowaniem banków bitom rejestru PORTB, procedury systemowe nie używają bezpośrednio numerów fizycznych banków, a numerów abstrakcyjnych. System wykonuje wykrywanie dostępnej pamięci operacyjnej podczas procedury inicjalizacji systemu, wpisując liczbę dostępnych banków do stałej systemowej BANKNUM\_ oraz wypełniając tablicę BANKTBL\_ ich numerami fizycznymi. Z wykorzystaniem tej tablicy wykonywane jest rozwiązywanie numerów abstrakcyjnych na wartości rejestru PORTB.

Mimo, iż system ma możliwość obsługi maksymalnej liczby 64 banków rozszerzonej pamięci, poprawną pracę zapewnia również w przypadku uruchomienia na niezmodyfikowanym komputerze Atari 65XE czy Atari 800XL, ze wyposażonym w fabrycznie zamontowane 64kB RAM.

### 5.2.2. Przydział pamięci dla procesów

Najprostszym sposobem przydziału pamięci dla procesów w przypadku komputerów XL/XE wyposażonych w dodatkową pamięć jest przydział każdemu procesowi osobnego

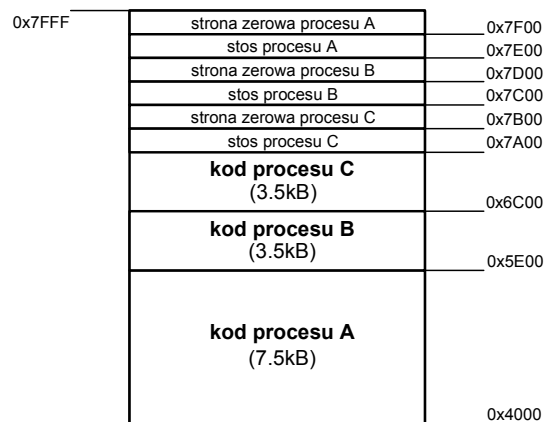
banku XMS. Przyjęcie tego modelu znakomicie upraszcza procedury alokujące zasoby, znosi również konieczność przygotowywania procesów jako kod relokowalny, gdyż każdy z procesów zostanie ulokowany w tym samym miejscu w przestrzeni adresowej. Rozwiązanie podobne zastosowane było w pierwszym systemie operacyjnym z podziałem czasu, CTSS<sup>35</sup>. Istotną wadą tego rozwiązania jest nieefektywne wykorzystanie dostępnej pamięci – niezależnie od długości kodu procesu zawsze zużyje on 16kB pamięci komputera, co w przypadku Atari 130XE powoduje, że nie uruchomimy jednocześnie więcej niż 5 procesów. Ten model przydziału pamięci został zaimplementowany z powodzeniem w prototypowej wersji systemu. Ze względu na konieczność przechowywania zawartości strony zerowej procesu oraz stosu dla programu użytkownika dostępne było 15,5kB pamięci:



Rys. 5.1. Organizacja pamięci banku w trybie przydziału pamięci „bank per proces”

Źródło: opracowanie własne

Modyfikacja tego sposobu przydziału pamięci opierała się na pomysłu podziału jednego 16kB banku na trzy nierówne części:



Rys. 5.2. Organizacja pamięci w trybie przydziału z wykorzystaniem podbanków

Źródło: opracowanie własne

<sup>35</sup> W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic

Wraz z zachowaniem sposobu przydziału pamięci „bank per proces” powstała możliwość alokacji pamięci dla trzech typów procesów:

- proces duży – o wielkości powyżej 7,5kB i poniżej 15,5kB,
- proces średni – o wielkości powyżej 3,5kB i poniżej 7,5kB,
- proces mały – o wielkości poniżej 3,5kB.

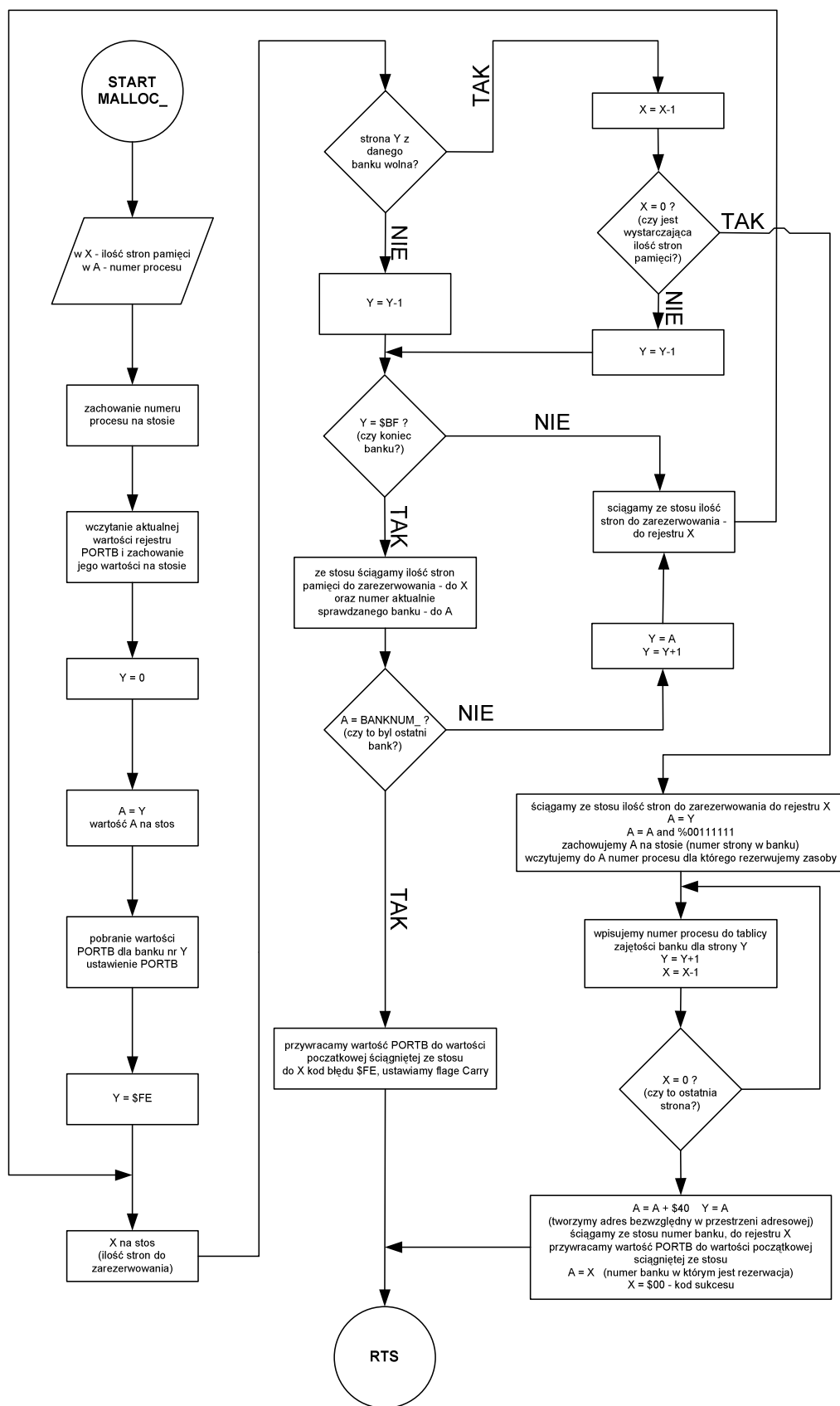
W ten sposób nawet na komputerze Atari 130XE możliwe było uruchomienie do 15 procesów. Dodatkowo ten model pozbawiony jest problemów związanych z fragmentacją pamięci, jaką znamy z metod liniowego przydziału.

Docelowym modelem przyjętym w ostatecznej wersji systemu jest przydział liniowy z dokładnością do jednej strony (256 bajtów). Procedura systemowa `MALLOC_`, odpowiedzialna za rezerwację pamięci, przeszukuje kolejno dostępne banki, poczynając od pamięci podstawowej – umożliwiając pracę systemu również na niezmodyfikowanym komputerze Atari 65XE bądź Atari 800XL – i wyszukując odpowiednie miejsce dla kodu procesu oraz trzech dodatkowych stron z przeznaczeniem na stronę zerową, stos procesu i dane kanałów IOCB. Algorytmem wykorzystywanym przy doborze miejsca jest algorytm pierwszego dopasowania (*ang. first-fit*)<sup>36</sup>: alokator porównuje wpisy dla poszczególnych stron, umieszczone w tablicy zajętości banku, rozpoczynając poszukiwania od strony 0x7E i zmniejszając licznik aż do osiągnięcia dolnej granicy przestrzeni adresowej danego banku. Jeżeli podczas przeszukiwania napotkany zostanie blok pamięci będący w stanie pomieścić proces, przeszukiwanie zostaje zakończone, odpowiednie strony oznaczone jako używane dla danego procesu i procedura zakańcza swoje działanie. Jeżeli dany bank nie posiada odpowiedniej ilości wolnego miejsca, przeszukiwanie według tego samego algorytmu rozpoczyna się dla kolejnego banku pamięci.

Szczegółowy algorytm działania procedury `MALLOC_`, z dokładnością do wykorzystywanych rejestrów i pojedynczych operacji, przedstawiony jest w formie diagramu blokowego na następnej stronie (Rys. 5.3).

---

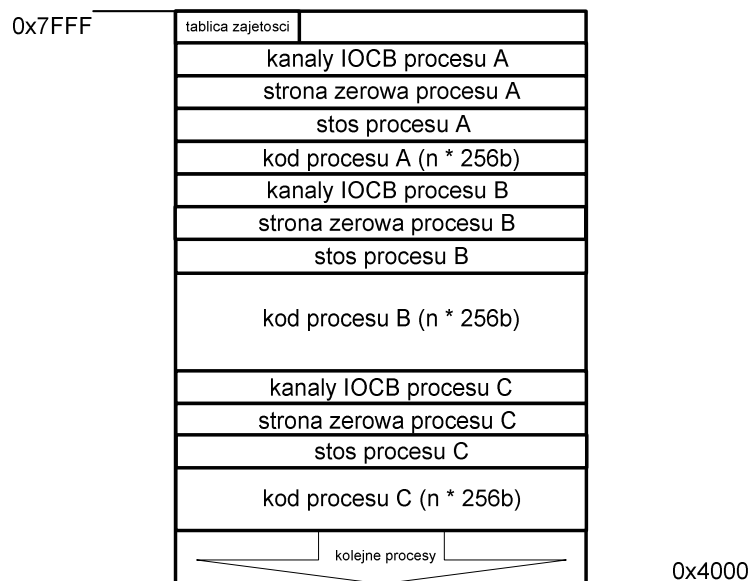
<sup>36</sup> W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic



Rys. 5.3. Schemat blokowy procedury przydziału pamięci MALLOC\_

Źródło: opracowanie własne

Finalna wersja organizacji pamięci banków XMS w trybie liniowej alokacji pamięci dla procesu zilustrowana jest poniżej:



Rys. 5.4. Organizacja pamięci banku XMS w trybie przydziału liniowego

Źródło: opracowanie własne

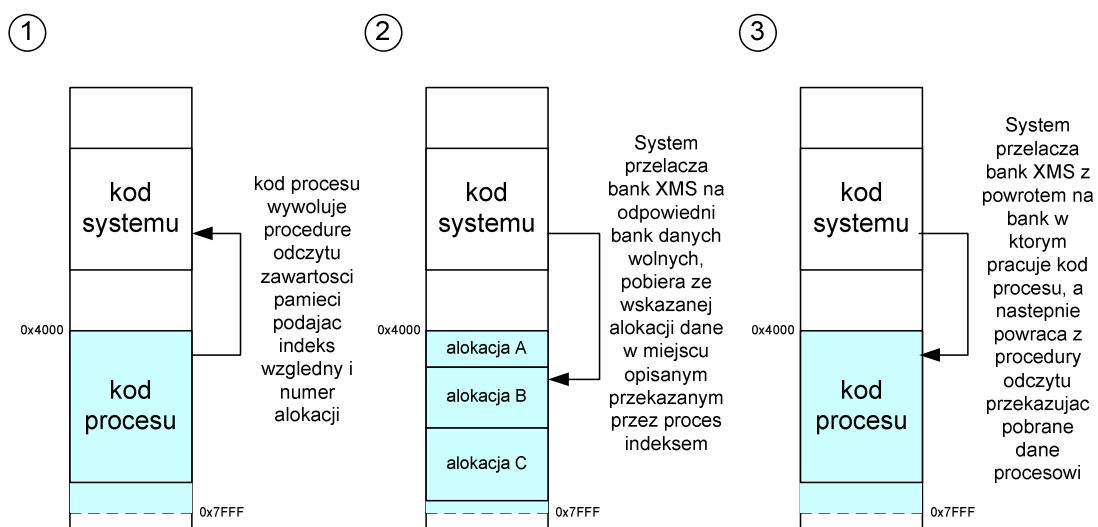
W sekcji opisanej na schemacie jako kod procesu znajdują się również segmenty danych szybkich, zarezerwowanych w procesie tworzenia pliku binarnego.

### 5.2.3. Sztuczne tryby adresowania

We wcześniejszej części niniejszego rozdziału opisano procedury przydziału pamięci podstawowej, alokowanej dla procesu podczas operacji jego wprowadzania. Rozmiar tej alokacji wyliczany jest na bazie długości segmentu kodu oraz zadeklarowanych przestrzeni segmentu danych szybkich, tj. danych, do których dostęp realizowany jest za pomocą naturalnych trybów adresowych procesora – nie może on jednak przekroczyć wielkości 15kB.

Realizacja przydziału większej ilości miejsca na dane dla procesu jest możliwa z wykorzystaniem tzw. sztucznych trybów adresowania, w którym operacje zapisu i odczytu z takiego segmentu danych – zwanego segmentem danych wolnych – wykonywane są za pośrednictwem procedur systemu operacyjnego. Proces po rozpoczęciu pracy zgłasza zapotrzebowanie do systemu operacyjnego na odpowiednią ilość stron pamięci, system

operacyjny w miarę dostępności dodatkowych banków pamięci przydziela pożądaną liczbę stron, zwracając procesowi numer alokacji, którym w przyszłości ten powinien posługiwać się chcąc zapisać bądź odczytać dane z zarezerwowanego obszaru. Miejsce w obszarze wskazywane jest jako indeks do początku obszaru danych, bez wskazania na konkretne miejsce w pamięci – wyliczeniem tego ostatniego zajmuje się już system operacyjny. Adres ten nie jest zresztą potrzebny procesowi, wykorzystanie go bez przełączenia banku XMS spowodowałoby dostęp do segmentu kodu własnego lub sąsiadującego procesu, natomiast próba samodzielnego przełączenia banku z dużym prawdopodobieństwem skutkowałaby zawieszeniem komputera, kiedy licznik rozkazu procesora trafiłby na losowe dane jakie pojawiłyby się w miejscu kodu. Procedura dostępu do danych wolnych przedstawiona jest na niniejszym schemacie:



Na diagramie kolorem oznaczona jest przestrzeń adresowa banku pamięci rozszerzonej XMS. Bezpośrednie odwołanie procesu do przestrzeni danych wolnych jest niemożliwe ze względu na wykorzystywanie tej samej przestrzeni adresowej.

Rys. 5.5. Procedura dostępu do danych wolnych

Źródło: opracowanie własne

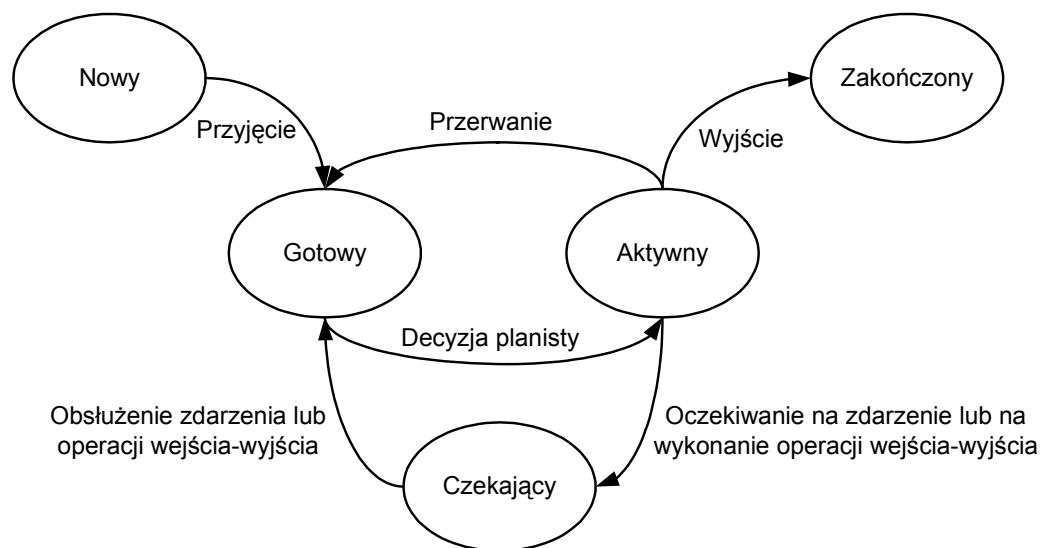
### 5.3. ZARZĄDZANIE PROCESAMI

Zasoby procesora są przydzielane poszczególnym procesom przez planistę krótkoterminowego (z ang. *short-term scheduler* lub *CPU scheduler*). W pierwotnie zaimplementowanym rozwiązaniu kod planisty uruchamiany był wyłącznie w takt przerwania VBLANK, czyli 50 razy na sekundę w przypadku systemów PAL. Planista odpowiedzialny jest za wybór kolejnego procesu do uruchomienia, przełączanie kontekstu oraz zarządzanie

pamięcią XMS. Ze względu na brak protekcji kodu jądra, również usuwanie procesów i zwalnianie wykorzystywanych przez nie zasobów wykonywane jest przez kod wywoływany przez planistę w przerwaniu. Mechanizm oparty na cyklicznym przerwaniu zapewnia wyłączenie procesu, niezależnie od tego, czym zajmuje się proces w danej chwili – dotyczy to także sytuacji procesu, który krąży w niekończącej się pętli. Ten sposób jest skuteczny, jednak nie do końca efektywny – zapewnia równy czas dla procesów o równych priorytetach niezależnie od ich potrzeb: proces dostaje czas procesora nawet wtedy, kiedy oczekuje na rezultat operacji wejścia/wyjścia. Aby zniwelować tę niedogodność, kolejna wersja jądra została wzbogacona o możliwość wywołania kodu planisty również po wystąpieniu programowego przerwania BRK. Takie wywołania zostały również umieszczone w procedurach wejścia/wyjścia – jeżeli dana procedura po wywołaniu przez proces nie będzie w stanie powrócić natychmiast do procesu z rezultatem operacji, zamiast oczekiwać w pętli oddaje systemowi zasoby niezwłocznie. Proces ten dokładniej opisany jest w dalszej części rozdziału.

### 5.3.1. Cykl życia i stany procesów

W tworzonym systemie zastosowany został zmodyfikowany klasyczny pięciostanowy model stanów procesów:

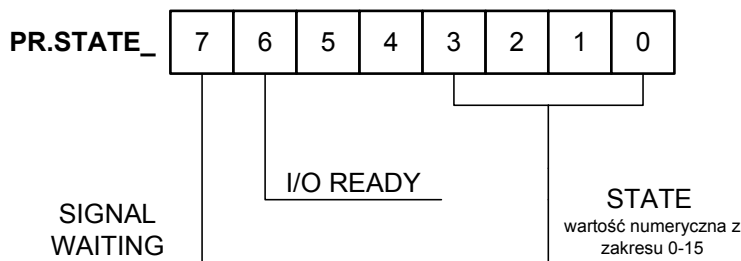


Rys. 5.6. Diagram stanów procesu

Źródło: A. Silberschatz, P. B. Galvin, *Podstawy systemów operacyjnych*, Warszawa 2002, Wydawnictwa Naukowo-Techniczne

Modyfikacje dotyczą podziału niektórych stanów na dwie składowe - pierwszy powstały w ten sposób „półstan” informuje planistę o konieczności obsługi procesu w sposób specjalny, drugi „półstan” jest potwierdzeniem zakończenia obsługi specjalnej i stanowi właściwy stan według powyższego diagramu. Dotyczy to stanów *Nowy* oraz *Czekający*, podzielone odpowiednio na *Tworzony* i *Nowy* oraz *Oczekujący na skolejkowanie* i *Czekający*. Specjalnym podtypem stanu *Czekający* jest dodatkowy stan *Zatrzymany*, w którym proces, zatrzymany przez SIGSTOP, oczekuje na sygnał wznowienia.

Stan procesu przechowywany jest w polu PR.STATE\_ strony bazowej. Znaczenie poszczególnych bitów przedstawione jest poniżej:



Rys. 5.7. Struktura pola PR.STATE\_

Źródło: opracowanie własne

Numer (bity 0-3)	Nazwa	Opis
\$00	<i>Nonexistent</i>	Proces nie istnieje – wolna pozycja w tablicy.
\$01	<i>Being created</i>	Ustawiany przez wczesną fazę procedury PRLOAD_, zanim sprawdzona zostanie dostępność pamięci.
\$02	<i>New</i>	Ustawiany przez środkową fazę procedury PRLOAD_, po zakończonej sukcesem rezerwacji zasobów dla procesu.
\$03	<i>Ready</i>	Proces gotowy do pracy.
\$04	<i>Running</i>	Proces aktualnie pracujący.
\$05	<i>Waiting for i/o</i>	Proces oczekuje na rezultat operacji wejścia/wyjścia.
\$06	<i>Terminating</i>	Proces zakończył swoje działanie, oczekuje na zwolnienie zasobów.
\$07	<i>To be queued for i/o</i>	Proces sygnalizuje planiście rozpoczęcie oczekiwania na rezultat operacji wejścia/wyjścia. Stan ten zastępowany jest stanem \$05 po zachowaniu kontekstu procesu.
\$08	<i>Stopped</i>	Proces zatrzymany (na skutek sygnału SIGSTOP).

Tabela 5.2. Znaczenie bitów 0-3 pola PR.STATE\_

Źródło: opracowanie własne



### 5.3.2. Uruchamianie procesów

Do dyspozycji procesu oddane zostały dwie funkcje wywołujące proces potomny. Pierwsza z nich, EXEC\_, uruchamia proces potomny wstrzymując pracę rodzica do czasu swojej terminacji. Potomek dziedziczy własność konsoli należącej do rodzica, a po zakończeniu swojej pracy może zwrócić rezultat jej działania rodzicowi w sposób analogiczny do przekazywania parametrów potomkowi (opisane szczegółowo w dalszej części tego rozdziału, w punkcie 5.1 – „Przekazywanie parametrów do procesu”). Terminacja potomka, dowolnie naturalna bądź wymuszona (np. z użyciem sygnału SIGKILL) powoduje automatyczne obudzenie procesu-rodzica i zwrócenie mu wszystkich zawłaszczonych wcześniej zasobów. Działanie procedury EXEC\_ nie jest tożsame z działaniem wywołania systemowego *exec()* znanego z systemów UNIX – w tamtym przypadku kod rodzica jest zamazywany przez kod nowo wprowadzanego procesu.

Drugą metodą wywołania procesu jest wywołanie do pracy współbieżnej z wykorzystaniem funkcji FORK\_. Funkcja ta wywołuje efekt zbliżony do techniki *fork-exec* znanej z UNIX. Technika ta powoduje wykonanie dokładnej kopii procesu, jak dla *fork()*, a następnie nadpisaniem jej przez kod potomka przez wywołanie *exec()*. W przypadku XEUX nowemu procesowi przydzielane są własne zasoby sprzętowe (numer konsoli TTY), a po zakończonej pracy planisty długoterminowego kontrola zwracana jest do procesu rodzica.

### 5.3.3. Planista długoterminowy

Cykl życia procesu rozpoczyna się z momentem pośredniego – poprzez EXEC\_ lub FORK\_ – wywołania przez proces rodzica procedury PRLOAD\_. Parametrami wyjściowymi są numer konsoli TTY dla procesu, lub sygnalizacja braku konsoli w postaci wartości \$FF (przekazywane w rejestrze Y), adres bufora do którego wczytany został plik binarny w relokowalnym formacie XX (odpowiednio LSB w akumulatorze i MSB w rejestrze X) oraz liczba i ewentualny adres przestrzeni parametrów, jakie należy przekazać tworzonemu procesowi (przekazywane odpowiednio w rejestrach PRLPRN\_ i PRLPAX\_).

Procedura rozpoczyna działanie od sprawdzenia, czy są wolne miejsca w tablicy stron bazowych procesu (w obecnej wersji systemu struktury danych zwymiarowane są pod maksymalnie 32 procesy). Po odnalezieniu miejsca niezwłocznie jest ono rezerwowane przez ustawienie dla danej pozycji stanu *Tworzony*.

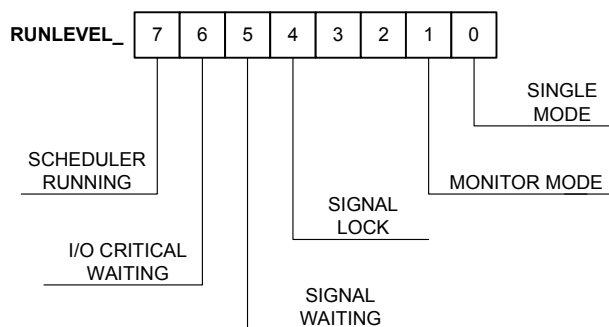
Następnie analizowany jest nagłówek pliku binarnego XX i na jego podstawie

określana długość procesu do załadowania. Wartość w bajtach przeliczana jest na liczbę stron, ta natomiast stanowi dane wejściowe dla opisaney już procedury `MALLOC_`. Jeżeli rezerwacja pamięci zakończyła się sukcesem, przepisywany jest tam kod procesu, a wpis stanu na stronie bazowej zmienia wartość na *Nowy*.

Następnie, na podstawie informacji dołączonych w bloku `fixupów`, dokonywana jest relokacja, czyli zwiększenie bardziej znaczącego bajtu argumentu wszystkich rozkazów odwołujących się do miejsca wewnątrz segmentu kodu lub danych szybkich procesu o indeks wynikający z adresu przyznanej przestrzeni. Jako kolejna inicjowana jest strona zerowa procesu, a następnie stos lokalny. Jeżeli podana przy wywoływaniu liczba parametrów jest większa od zera, parametry oraz ich liczba umieszczane są na stosie. W przeciwnym przypadku stos stanowić będzie pojedyncza wartość 0, oznaczająca brak parametrów. Na zakończenie inicjowana jest strona bazowa procesu oraz wektory obsługi sygnałów, po czym stan procesu zmienia się na *Gotowy*, co zakańcza wprowadzanie procesu.

#### 5.3.4. Planista krótkoterminowy

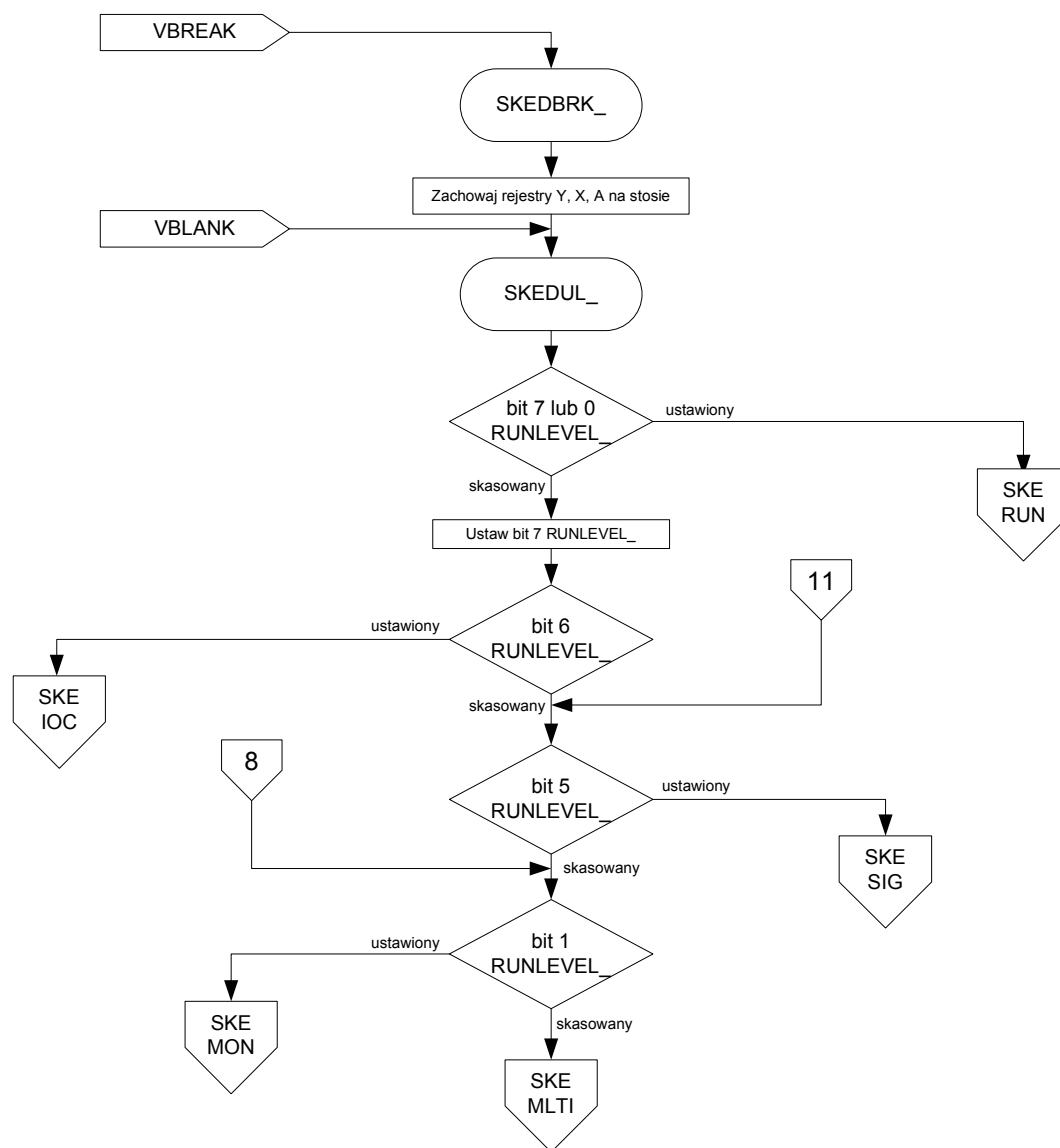
Kod planisty (ang. *scheduler*), uruchamiany periodycznie za każdym wystąpieniem przerwania VBL, a również na żądanie procedur systemowych po wystąpieniu przerwania BRK, stanowi właściwe jądro systemu. Składa się z kilku części, odpowiedzialnych m.in. za kontrolę czasu procesora zużywanego przez procesy i odpowiednie ich wywłaszczanie, przełączanie kontekstu procesów, obsługę sygnałów, wywoływanie procesów, które oczekują na krytyczną czasowo operację wejścia/wyjścia, obsługę trybu monitora systemowego, czy wreszcie terminację procesów i zwalnianie zajmowanych przez nie zasobów. Do sygnalizacji potrzeby uruchomienia danej funkcji planisty wykorzystujemy bity rejestru `RUNLEVEL_`:



Rys. 5.8. Znaczenie bitów pola `RUNLEVEL_`

Źródło: opracowanie własne

Pierwotnie rejestr RUNLEVEL\_, tak jak i pole PR.STATE\_, przyjmował wartości liczbowe z zakresu 0-255, ze znaczeniem przypisanym dla poszczególnych wartości. Koncepcja ta została jednakże zmodyfikowana ze względu na większą elastyczność rozwiązania opartego na pojedynczych bitach znaczących. Od testowania bitów tego rejestru rozpoczyna się kod planisty:

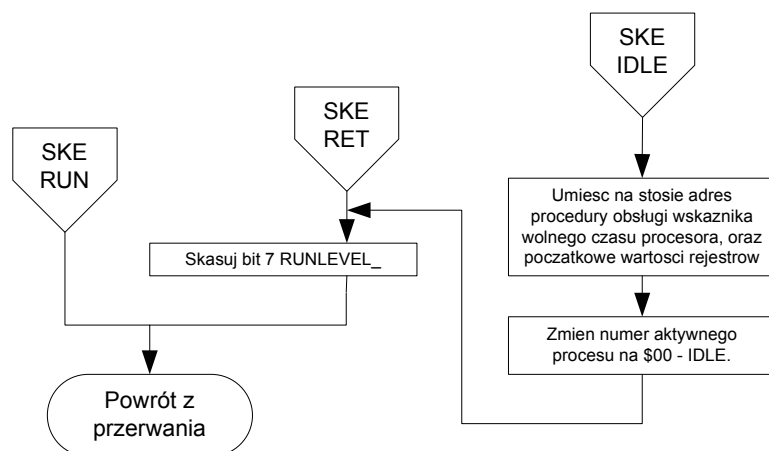


Rys. 5.9. Schemat blokowy planisty krótkoterminowego – procedura główna

Źródło: opracowanie własne

Wejście do głównej procedury różni się w zależności od typu przerwania – w przypadku wektora VBREAK zanim rozpocznie się główny kod, rejestry Y, X i A muszą być zachowane na stosie. Kod planisty uruchamiany jest nie tylko cyklicznie co 1/50 sekundy, ale również na skutek wystąpienia przerwania programowego, które wystąpić może w dowolnym

czasie. Z tego powodu niezbędne jest zabezpieczenie przed wywołaniem kodu planisty przez przerwanie VBLANK z wewnątrz już działającego planisty, uruchomionego programowo – służy do tego sprawdzenie bitu 7 `RUNLEVEL_`, którego ustawienie sygnalizuje wykonywanie kodu systemowego. W przypadku wykrycia takiej sytuacji, a również w sytuacji, gdy ustawiony jest bit wskazujący na jednoprocesowy tryb pracy, wykonywany jest skok do drugiego wejścia procedury `SIG_RET_` nazwany `SIG_RUN_`:



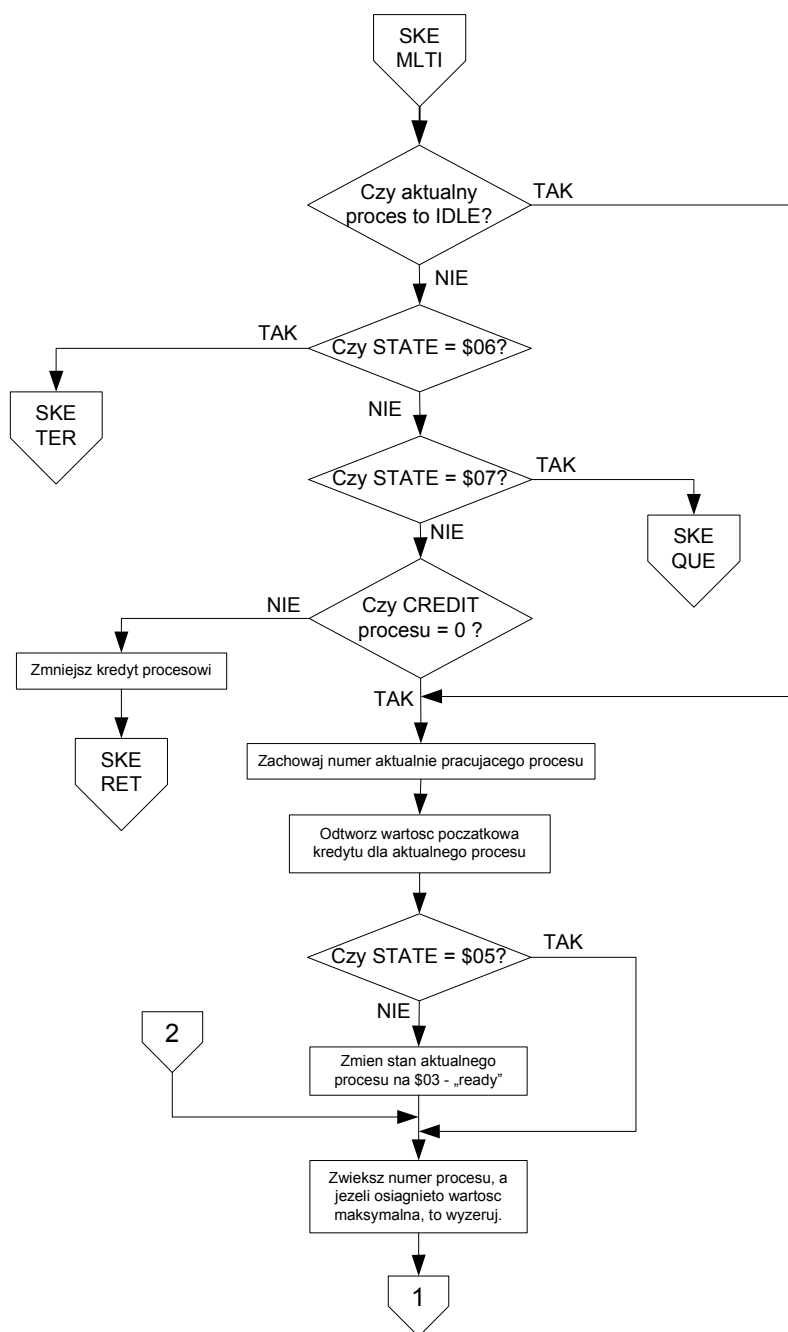
Rys. 5.10. Schemat blokowy planisty krótkoterminowego – procedury powrotu

Źródło: opracowanie własne

Częścią tej sekcji kodu planisty jest również kod przekazujący kontrolę do metaprocesu beczynności (z ang.: *idle*), który jako jedyny proces systemowy zawsze jest w jednym z dwóch stanów – *Ready* lub *Running*. Metaproces ten nie jest pełnoprawnym procesem, co widoczne będzie wyraźnie podczas analizy dalszej części programu jądra – podczas jego wywłaszczania nie zachowywane są rejestry, stos, strona zerowa ani nawet licznik programu. Wykonywanie procesu beczynności nigdy nie trwa całego kwantu czasu pomiędzy przerwaniem VBLANK, po kilkudziesięciu cyklach wywoływane jest przerwanie programowe BRK i kontrola oddawana jest z powrotem do planisty, który sprawdza, czy nie pojawiły się inne procesy w stanie gotowym do uruchomienia. Ze względu na uproszczoną procedurę przełączania kontekstu IDLE, nie ma potrzeby wyłączania jego obsługi nawet w przypadku, gdy w systemie pracują inne procesy gotowe do wykonywania.

Przetwarzaniem kolejki procesów oraz podejmowaniem decyzji dotyczących ich uruchomienia bądź wywłaszczenia zajmuje się sekcja `SKE_MLTI_` planisty krótkoterminowego. Ze względu na czytelność, algorytm jej działania został przedstawiony w formie dwóch dopełniających się schematów blokowych.

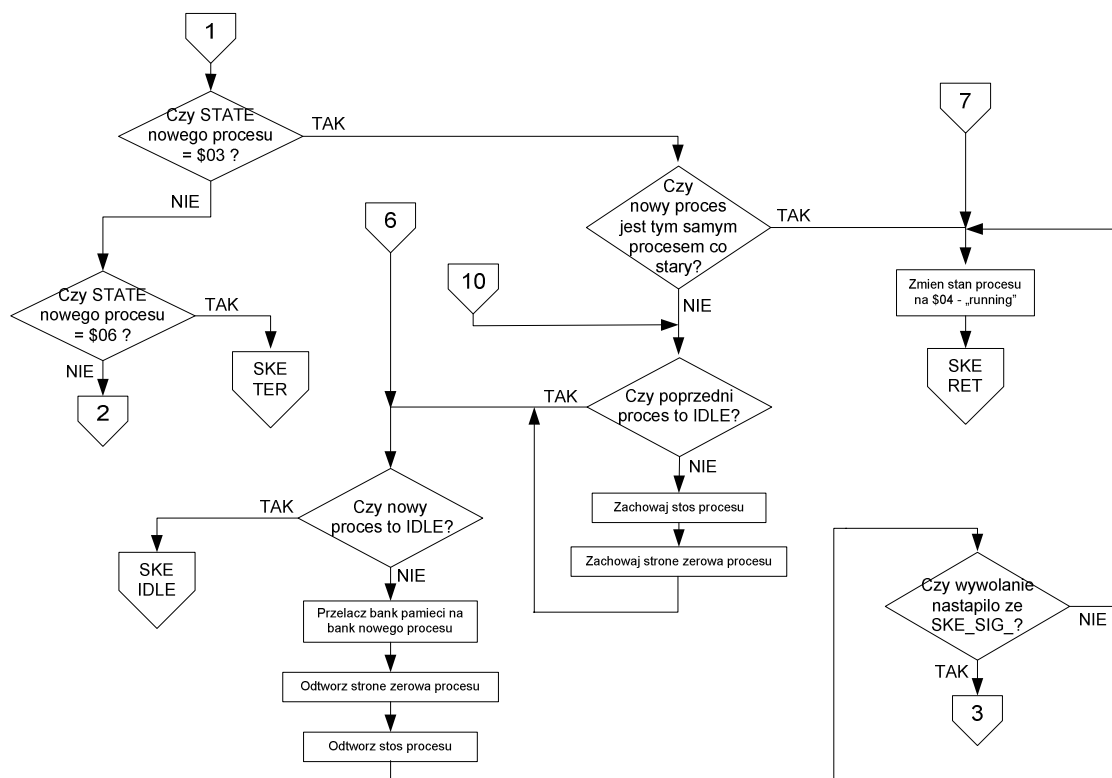
Pierwszy z nich to fragment analizujący stan bieżącego procesu – czy proces nie jest procesem beczynności, czy nie zakończył działania, oraz czy nie oddaje zasobów ze względu na oczekiwanie na rezultat operacji wejścia/wyjścia. W tym też miejscu sprawdzany jest jego kredyt – jeżeli nie zachodzą żadne z powyżej wymienionych warunków, a kredyt procesu jest dodatni, planista kończy pracę i zwraca kontrolę przerwanemu procesowi.



Rys. 5.11. Schemat blokowy planisty krótkoterminowego – SKE\_MLTI\_ część 1

Źródło: opracowanie własne

Mechanizm kredytu procesu ma za zadanie ograniczyć częstość przełączania kontekstu, tak aby nie zużywać większości czasu procesora na czynności administracyjne. Domyślne ustawienia to przełączenie procesu co dwa wywołania planisty. Wartość ta może być różna dla każdego z procesów istniejącego w systemie.



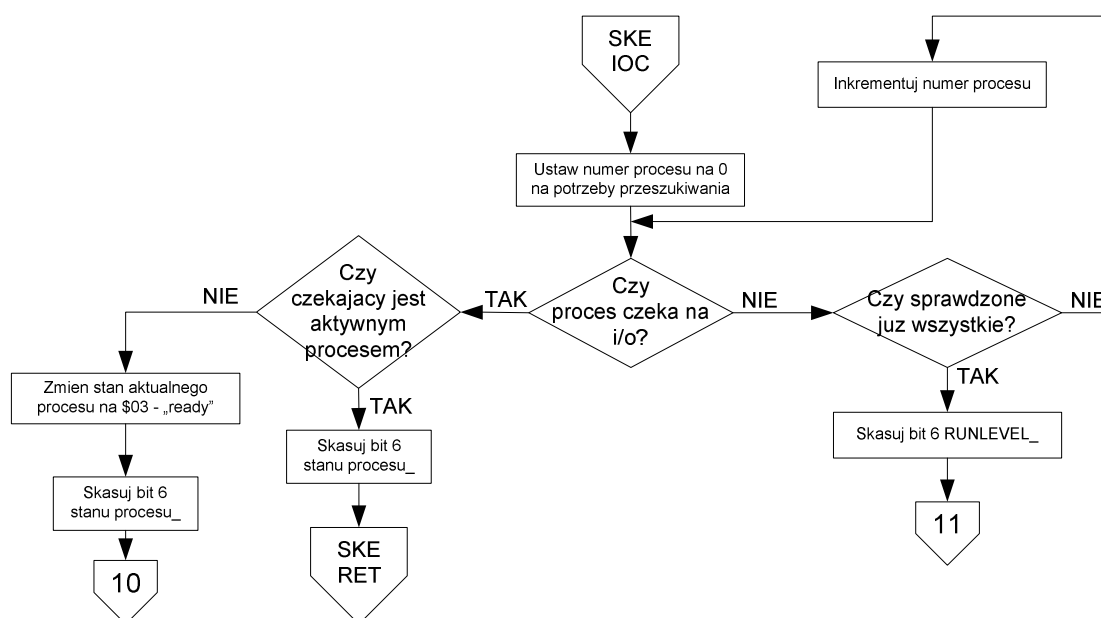
Rys. 5.12. Schemat blokowy planisty krótkoterminowego – SKE\_MLTI\_ część 2

Źródło: opracowanie własne

Rysunek 5.12 prezentuje algorytm działania części planisty wykonującego przełączenie kontekstu procesu. Sekcja ta oznaczona jest w kodzie źródłowym etykietami ?STORE oraz ?SWITCH i jak widać po ilości odwołań do innych sekcji schematu jest często wykorzystywana w całości lub w częściach przez inne fragmenty kodu planisty.

Wielokrotne wykorzystywanie kodu o identycznych funkcjach jest trudniejsze w przypadku kodu planisty niż w przypadku kodu systemowego wywoływanego przez procesy – jako że planista wykonuje operacje na stosie, zachowując go i odtwarzając dla różnych procesów, nie jest możliwe używanie, korzystającego ze stosu do przechowywania adresu powrotu, rozkazu skoku do procedury JSR (z ang.: *Jump to SubRoutine*).

Następnym prezentowanym fragmentem kodu jądra jest krótka procedura planisty wywołująca proces oczekujący na rezultat operacji wejścia/wyjścia:



Rys. 5.13. Schemat blokowy planisty krótkoterminowego – SKE\_IOC\_

Źródło: opracowanie własne

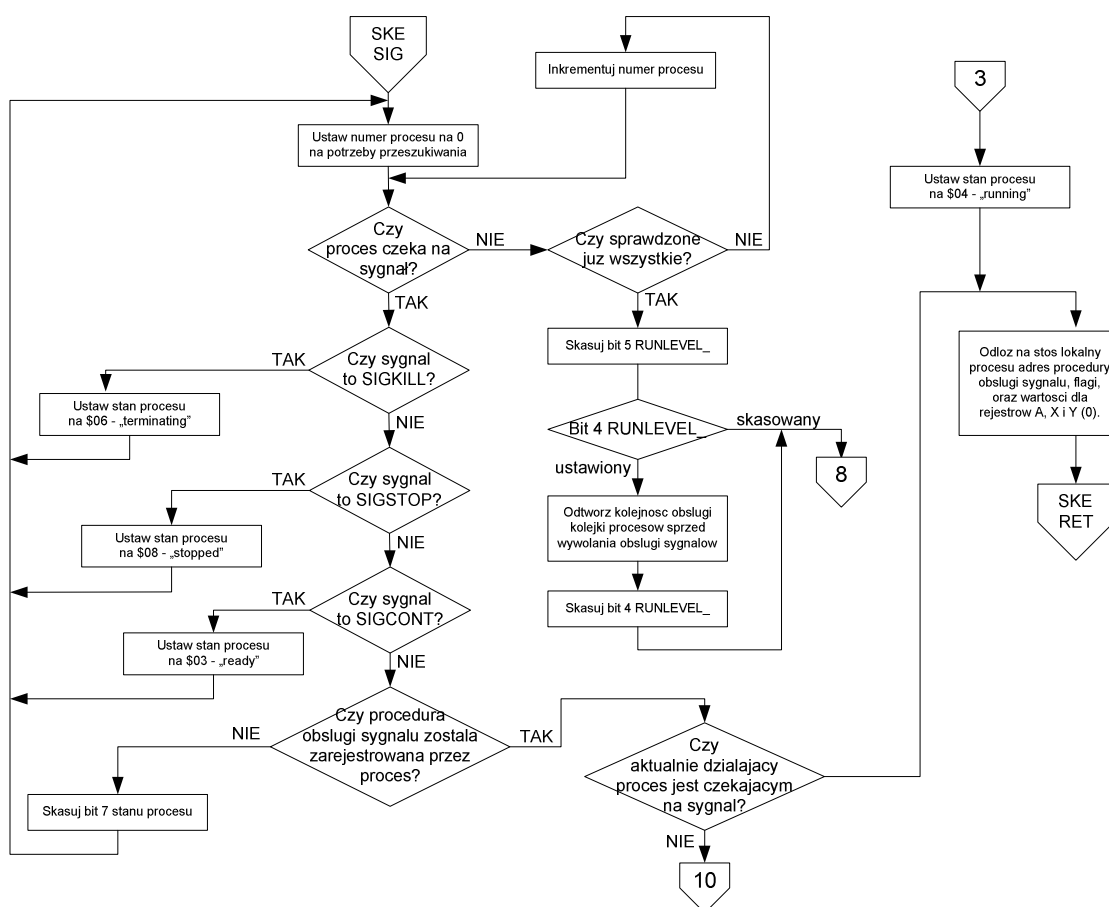
### 5.3.5. Przerwania systemowe w środowisku wieloprotocowym

Obsługa przerwania procesora w systemie XEUX jest wykonywana wyłącznie przez procedury systemowe. Jest to rozwiązanie konieczne, gdyż funkcjonowanie systemu jako całości zależy od poprawnego wywoływania jego istotnych części, takich jak planista krótkoterminowy czy procedury obsługi urządzeń wejścia/wyjścia. Procedury te powinny zostać uruchomione w odpowiednich momentach przez procedury obsługi przerwania, a ingerujący w nie kod programu użytkownika mógłby wywołania te uniemożliwić lub zaburzyć.

Drugim istotnym aspektem wymuszającym tego typu rozwiązanie jest fakt, iż w środowisku wieloprotocowym kontekst procesu ulega częstym przełączeniom – dotyczy to zarówno zawartości stosu, strony zerowej, ale także istnienia samego kodu procesu w danej przestrzeni adresowej. Zainstalowany poprawnie wektor przerwania, wskazujący na procedurę jego obsługi w kodzie programu użytkownika, po przełączeniu banku, mógłby wskazywać w najlepszym wypadku środek kodu innego procesu, a w bardziej prawdopodobnym fragmencie danych lub argument operacji, który zinterpretowany przez

procesor jako instrukcja wywołałby efekty trudne do przewidzenia, prawdopodobnie w krótkim czasie zawieszając komputer.

W miejsce utraconych przerw procesy zyskują mechanizm obsługi sygnałów, dający podobnie jak w przypadku przerwania możliwość zarejestrowania własnych procedur obsługi. Wystąpienie sygnału obsługiwane jest przez planistę w sposób priorytetowy, udzielając procesowi, dla którego oczekuje sygnał, dodatkowego czasu ponad przydzieloną kolejkę. Obsługa sygnałów zwykłych w priorytecie ustępuje jedynie obsłudze niejawnych sygnałów procesora, wykorzystywanych do budzenia procesów oczekujących na rezultat krytycznych czasowo operacji wejścia/wyjścia.



Rys. 5.14. Schemat blokowy planisty krótkoterminowego – SKE\_SIG\_

Źródło: opracowanie własne

Powyższy diagram to schemat funkcjonowania fragmentu kodu planisty odpowiedzialnego za obsługę sygnałów. Wywoływany jest z procedury głównej, a priorytet jej wywołania jest niższy tylko od procedury obsługi krytycznej czasowo operacji wejścia/wyjścia.



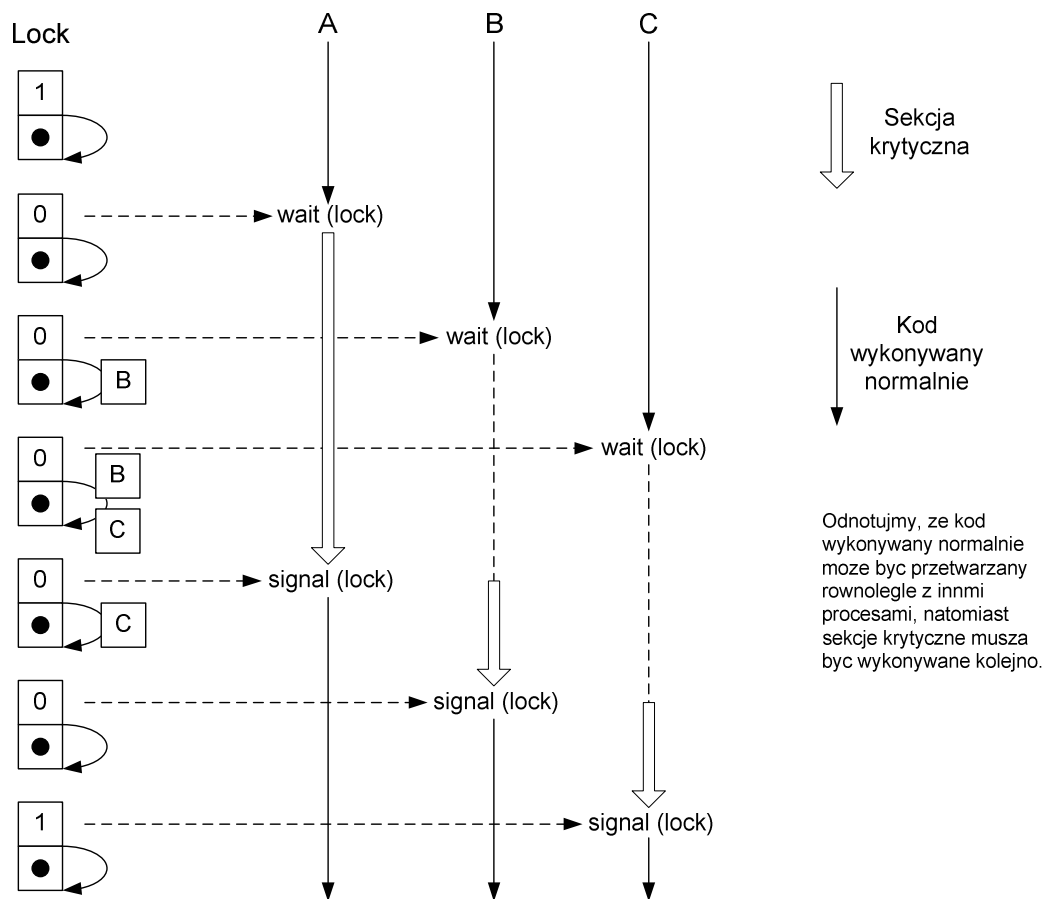
### 5.3.6. Semafor

W celu uniknięcia konfliktu w dostępie do wspólnych zasobów (np. konsola systemowa, bądź bufor wymiany w operacji wczytywania procesu potomnego) zaimplementowany został mechanizm semaforów binarnych. Ich funkcjonowanie omówione zostanie szczegółowo na przykładzie drugiego wspomnianego zastosowania, tj. współdzielenia bufora wczytywania procesu potomnego.

Bufor służący do pobrania nowego procesu z pamięci masowej powinien być tak duży, jak wielkość pliku binarnego, aby możliwa była analiza wszystkich opcjonalnych nagłówków przed rozpoczęciem operacji alokowania zasobów. Dodatkowo powinien on znajdować się w obszarze pamięci poza przestrzenią adresową bloku XMS – w ten sposób możliwe będzie bezproblemowe kopiowanie jego zawartości w docelowe miejsce zaalokowane dla procesu w jednym z banków XMS. Z tych przyczyn wynika, że nie jest możliwe do zrealizowania – przynajmniej bez wykorzystania wolnych, sztucznych trybów adresowania, co mogłoby okazać się rozwiązaniem mało efektywnym – utworzenie osobnego bufora dla każdego chcącego go wykorzystywać procesu.

Rozwiązaniem jest zastosowanie jednego bufora ulokowanego w obszarze pamięci głównej systemu, zapewniając jednakże możliwość bezkolizyjnej współpracy wielu wykorzystujących tę wspólną przestrzeń procesów.

Rysunek 5.15 prezentuje funkcjonowanie mechanizmu semaforów na przykładzie trzech procesów (A, B i C) korzystających z tego samego zasobu zabezpieczonego semaforem *lock*. Wywołanie przez proces A operacji *wait(lock)* powoduje sprawdzenie wartości semafora – ponieważ jest on ustawiony, sekcja krytyczna procesu może się rozpocząć. Przed przekazaniem kontroli do sekcji krytycznej, semafor jest kasowany. Następny proces wykonujący *wait(lock)* zostanie wstrzymany aż do czasu ponownego ustawienia semafora, co nastąpi dopiero po wykonaniu przez A operacji *signal(lock)* – czyli dopiero po zakończeniu wykonywania jego sekcji krytycznej.



Rys. 5.15. Procesy wykorzystujące współdzielone dane zabezpieczone semaforem

Źródło: W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic

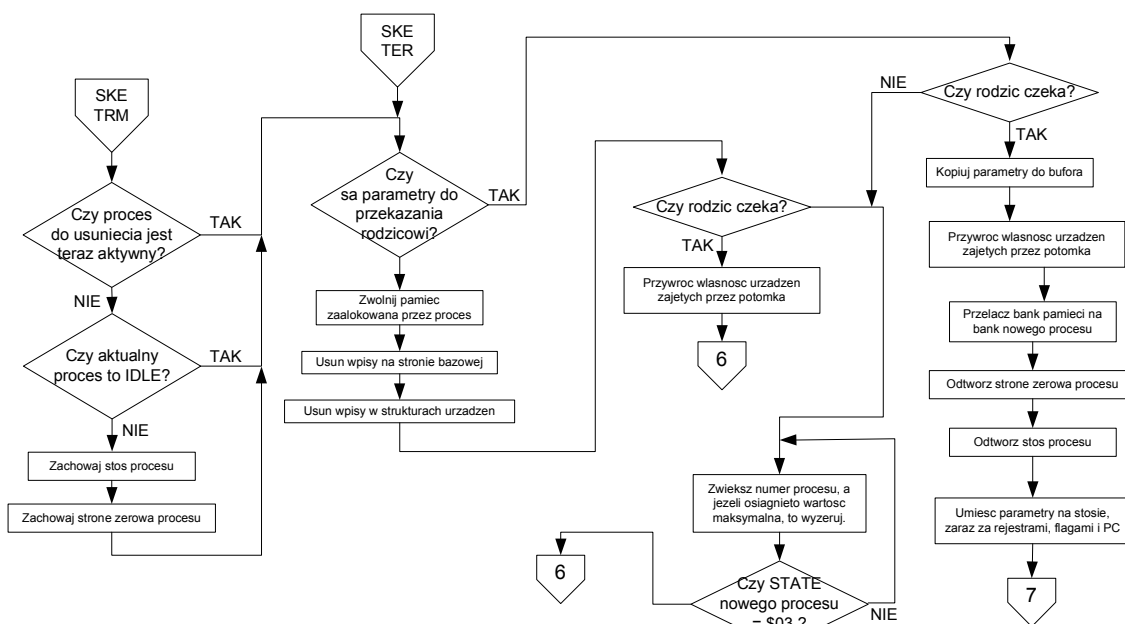
Procedurą wykorzystującą opisany na początku przykładu bufor jest `PRREAD_`, wczytująca z dysku plik binarny procesu przeznaczonego do uruchomienia. Na początku swojego działania wykonuje operację `wait` na semaforze `PRREADL_` - jeżeli wartość wynosi `$FF`, oznacza to, że zasób jest wolny i po wpisaniu wartości numeru procesu do semafora kontrola oddawana jest do procedury celem umożliwienia wczytania pliku. Wartość inna niż `$FF` powoduje wprowadzenie procesu do kolejki oczekujących i efektywne przełączenie kontekstu na następny gotowy do uruchomienia proces.

Po zakończeniu pracy z wykorzystaniem bufora, tj. po skutecznym wprowadzeniu procesu do systemu, następuje wywołanie operacji `signal` dla `PRREADL_`, co powoduje usunięcie wpisu w semaforze i sprawdzenie, czy na zmianę jego wartości oczekują inne procesy. Jeżeli tak, jeden z nich jest budzony, a jego numer wprowadzany do semafora. Jeżeli żaden z procesów nie oczekuje, do semafora wprowadzana jest wartość początkowa `$FF`.

Zaimplementowany model różni się od modelu teoretycznego szczególnie wpisywania do semafora numeru procesu, jaki wykonuje obecnie sekcję krytyczną z wykorzystaniem

### 5.3.7. Terminacja procesu

Algorytm działania fragmentu jądra systemu odpowiedzialnego za terminację procesów ilustruje poniższy diagram:



Źródło: opracowanie własne

Jeżeli proces kończący pracę chce przekazać swojemu rodzicowi dane (np. stanowiące produkt jego działania), wykonuje to korzystając z podobnego mechanizmu, jak w przypadku przekazywania parametrów do procesu – podając liczbę bajtów parametrów do rejestru PRLPRN\_, a adres początku obszaru danych do PRLPAX\_. Dla zachowania kompatybilności z plikami binarnymi wytworzonymi zanim pojawiła się ta funkcjonalność wyjście z procesu należy wykonać przez skok do procedury TERMWDATA\_ zamiast TERMINATE\_ - ta druga przyjmuje domyślnie brak przekazywanych parametrów.

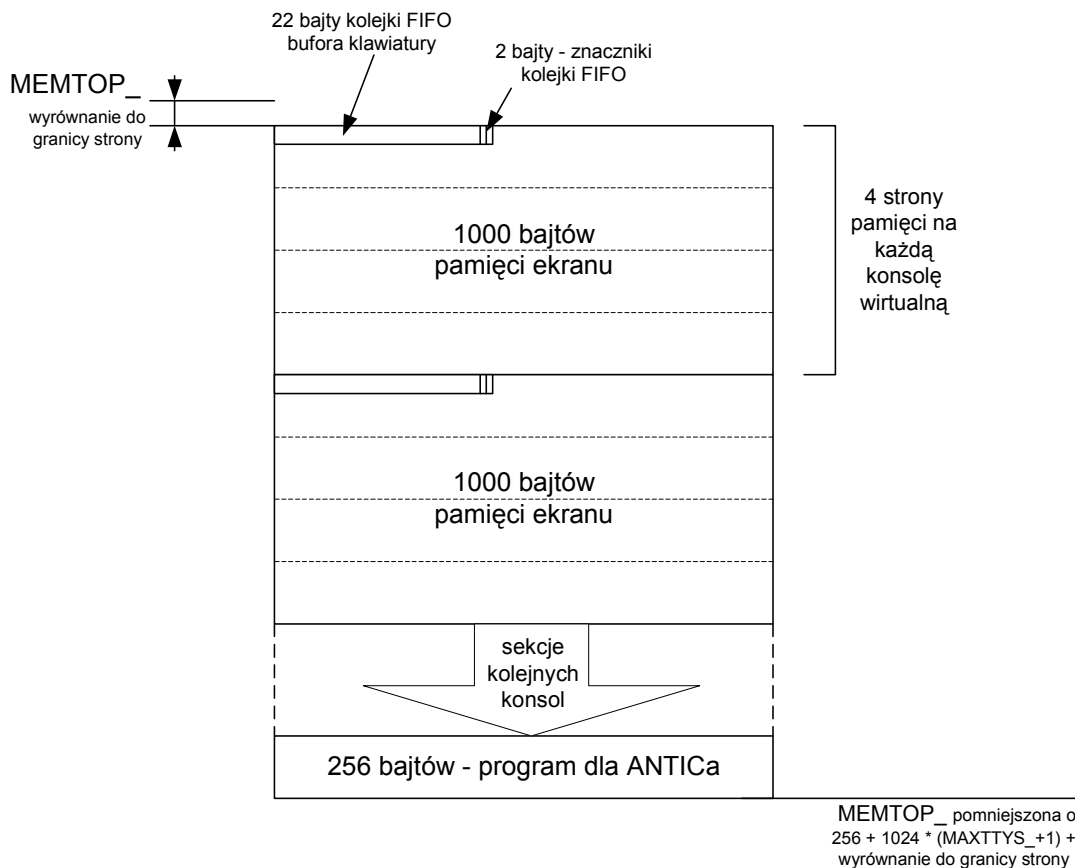
## **5.4. STEROWNIKI URZĄDZEŃ**

### **5.4.1. Konsole wirtualne**

System XEUX został wyposażony w mechanizm wirtualnych konsol, znany z obecnych implementacji systemów UNIX. Każda z nich umożliwia niezależne korzystanie z ekranu oraz klawiatury przypisanemu do niej procesowi. Użytkownik może w dowolnym momencie przełączać się pomiędzy konsolami za pomocą zdefiniowanej w systemie kombinacji klawiszy (CTRL+SHIFT+numer konsoli). W sterowniku zostały zaimplementowane wybrane funkcje systemowych urządzeń K:, S: oraz E: Atari OS, dodany 22 bajtowy bufor klawiatury, a rozmiar ekranu powiększony z tradycyjnych 24 do częściej spotykanych na innych platformach 25 linii. Konsole domyślnie przyjmują równej wielkości dwuznakowe marginesy po obu stronach ekranu, jednakże dowolny proces może dostosować je do swoich potrzeb, modyfikując wartości w komórkach LMARGN\_ oraz RMARGN\_.

Domyślna liczba konsol wirtualnych to 4 – wartość ta zdefiniowana jest w sekcji stałych systemowych jako MAXTTYS\_ i można ją zwiększyć według potrzeb, pamiętając jedynie o tym, że każda dodatkowa konsola wymaga dodatkowej zaalokowanej na te potrzeby pamięci i odpowiednio poszerzonych struktur przechowujących informacje o pozycji kursora, marginesach czy przypisaniu do procesu. Pamięć ta alokowana jest od góry, a jako górna granica obszaru wolnej pamięci przyjmowana jest wartość aktualizowanej w fazie inicjalizacji systemu zmiennej MEMTOP\_, zaokrąglona do pełnej strony pamięci.

Sposób organizacji pamięci dla systemu wirtualnych konsol prezentuje Rysunek 5.17:



Rys. 5.17. Organizacja pamięci dla konsol wirtualnych w XEUX

Źródło: opracowanie własne

Wszystkie konsole współdzielą ten sam program dla procesora graficznego, generowany dynamicznie przez procedurę TTYINI\_ (TTY INIT):

```

$70 ; rozkaz generowania 8 pustych linii
$70 ; kolejne 8 linii
$42 ; rozkaz LMS (Load Memory SCAN) +
    ; 1 linia trybu tekstowego GR.0
$00 ; LSB początku pamięci ekranu
$xx ; MSB początku pamięci ekranu
$02 ; 1 linia trybu tekstowego GR.0
...
... ; komenda $02 powtórzona 24 razy
...
$41 ; rozkaz JVB (Jump and wait for VBlank)
$00 ; LSB początku programu Antica
$yy ; MSB początku programu Antica

```

W miejsce \$yy procedura TTYINI\_ wpisuje adres strony, na której buduje powyższy program, natomiast wartość \$xx pozostawiana jest wyzerowana. Wypełnia ją procedura

TTYSWT\_ (TTY SWiTch), wpisując aktualny adres pamięci ekranu podczas przełączania aktywnej konsoli. Adres ten pobierany jest z tablicy TTYTBL\_ (TTY TaBLe), budowanej w początkowej fazie działania procedury TTYINI\_.

Procedura TTYSWT\_ uaktywnia daną konsolę, tj. powoduje iż ekran komputera oraz jego klawiatura przełączane są na jej potrzeby – to widzi użytkownik. Proces odwołuje się do zawartości ekranu oraz bufora klawiatury korzystając z ujednoliconego podsystemu wejścia/wyjścia XXCIO (*Xeux eXtended Central Input Output*). Procedury bezpośrednio odpowiedzialne za obsługę wirtualnych konsol, wywoływane wewnętrznie przez XXCIO, korzystają ze zmiennych oraz wektorów które uaktualniane są dla każdego procesu przez planistę podczas przełączania kontekstu na podstawie informacji zawartych na stronie bazowej. Za uaktualnienie danych odpowiedzialna jest procedura PTTYSWT\_ (Process's TTY SWiTch), używającą tablic TTYTBL\_ oraz TSTTBL\_ (Tty SeTtings TaBLe) jako struktur do przechowywania danych charakteryzujących daną konsolę:

Pozycja	Etykieta	Funkcja
1	MXCOLT_	Zawartość zmiennej MAXCOL_, przechowującej liczbę kolumn konsoli
2	MXROWT_	Zawartość zmiennej MAXROW_, przechowującej liczbę wierszy konsoli
3	CLCRST_	Zawartość zmiennej COLCRS_, przechowującej pozycję poziomą kursora
4	RWCRST_	Zawartość zmiennej ROWCRS_, przechowującej pozycję pionową kursora
5	LMRGNT_	Zawartość zmiennej LMARGN_, przechowującej wartość lewego marginesu
6	RMRGNT_	Zawartość zmiennej RMARGN_, przechowującej wartość prawego marginesu

Tabela 5.3. Zawartość tablicy TSTTBL\_

Źródło: opracowanie własne

Liczba pozycji danego typu odpowiada ilości konsoli wirtualnych w systemie, domyślnie tablica zwymiarowana jest pod 4 konsole.

Kody naciskanych przez użytkownika klawiszy trafiają do 22 bajtowych kolejek FIFO wygospodarowanych na końcu czwartej strony pamięci każdej konsoli, zaraz za 1000 bajtowym obszarem przechowującym zawartość ekranu. Wywołana przez użytkownika procedura XXCIO odczytująca znak z konsoli pobiera wartość z bufora za pomocą procedury KGETCH\_ (z ang.: *Keyboard: GET Character*). Otrzymany jako rezultat jej działania kod to już wartość znaku skonwertowana przedstawiona w kodzie ATASCII:

## ATASCII

00 ♥	10 ♠	20	30 0	40 0	50 P	60 ♣	70 P
01 ♠	11 ♠	21 !	31 1	41 A	51 Q	61 a	71 q
02	12 —	22 "	32 2	42 B	52 R	62 b	72 r
03 J	13 +	23 #	33 3	43 C	53 S	63 c	73 s
04 †	14 ●	24 \$	34 4	44 D	54 T	64 d	74 t
05 †	15 ■	25 %	35 5	45 E	55 U	65 e	75 u
06 /	16	26 &	36 6	46 F	56 V	66 f	76 v
07 \	17 T	27 '	37 7	47 G	57 W	67 g	77 w
08 ▲	18 L	28 €	38 8	48 H	58 X	68 h	78 x
09 ■	19 ■	29 )	39 9	49 I	59 Y	69 i	79 y
0A ▲	1A L	2A *	3A :	4A J	5A Z	6A j	7A z
0B ■	1B €	2B +	3B ;	4B K	5B [	6B k	7B ♣
0C ■	1C †	2C ,	3C <	4C L	5C \	6C l	7C
0D —	1D †	2D —	3D =	4D M	5D J	6D m	7D K
0E —	1E †	2E .	3E >	4E N	5E ^	6E n	7E †
0F ■	1F †	2F /	3F ?	4F O	5F —	6F o	7F ♠

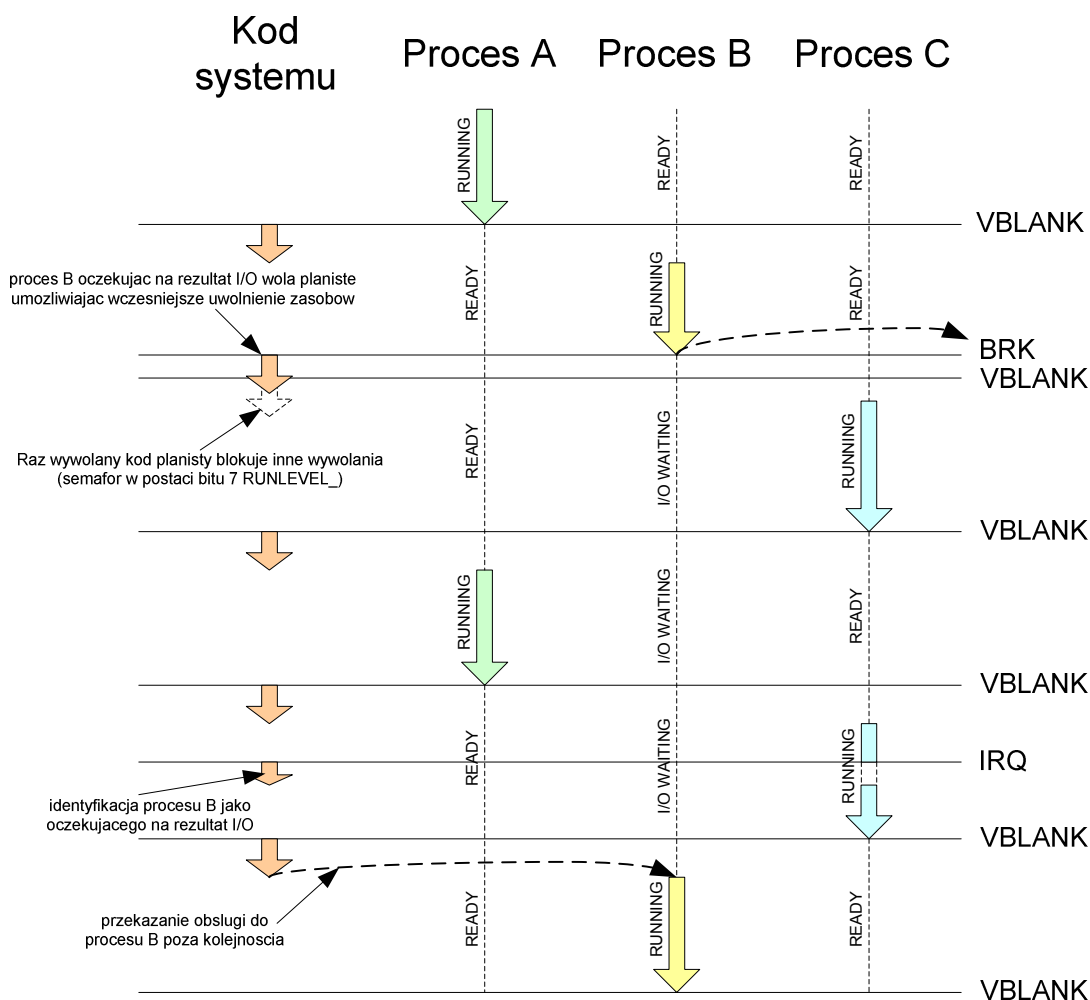
Rys. 5.18. Tablica kodów ATASCII podstawowego zestawu znaków

Źródło: <http://atariki.krap.pl/index.php/ATASCII>

Jeżeli podczas zgłoszenia żądania odczytu bufor jest pusty, procedura KGETCH\_ zmienia stan aktualnego procesu na \$07 – „*to be queued for i/o*”, zapisuje na stronie bazowej rodzaj informacji wejścia/wyjścia na rezultat której proces oczekuje, a następnie niezwłocznie wzywa planistę krótkoterminowego, aby odebrał procesowi zasoby CPU. Planista po otrzymaniu informacji zachowuje kontekst oczekującego procesu na stronie bazowej, potwierdzając ten fakt zmianą stanu procesu na \$05 – „*waiting for i/o*”. Dzięki temu procesy nie marnują czasu procesora na krążenie w pętli w oczekiwaniu na pojawienie się rezultatu, a czas ten może być wykorzystany przez inne procesy.

Naciśnięcie klawisza przez użytkownika powoduje wywołanie procedury PUTKBUF\_ (z ang.: *PUT the Key in the BUFfer*), wpisującej kod klawisza (będący na tym etapie wewnętrznym kodem klawiatury) do bufora tej konsoli, która widoczna jest w momencie jego wciśnięcia. Po wprowadzeniu klawisza sprawdzane jest przypisanie w tablicy TTYOWNT\_ (z ang.: *TTY OWNership Table*) wskazujące na proces aktualnie używający danej konsoli, a po upewnieniu się, że oczekuje on na dane właśnie z niej (stan \$05 – „*waiting for i/o*” oraz wpis w PR.IOQUE\_ wskazujący tą konsolę wirtualną), ustawiany jest bit 6 statusu procesu – sygnalizując, że dla procesu oczekuje rezultat operacji i/o, oraz bit 6 RUNLEVEL\_, który ma za zadanie poinformować planistę o konieczności priorytetowej obsługi zdarzenia wejścia/wyjścia. Podczas najbliższego uruchomienia kodu planisty (na skutek wystąpienia cyklicznego przerwania VBLANK, lub programowego BRK – czyli w najgorszym wypadku po niecałej 1/50 sekundy), w pierwszej kolejności obsłuży on proces oczekujący, następnie

powróci do przetwarzania kolejki procesów w dotychczasowym porządku. Czas przyznany na obsługę operacji wejścia/wyjścia jest dodatkowym czasem przydzielanym procesowi i nie powoduje jego wypadnięcia z kolejki przydziału procesora.



Rys. 5.19. Model uproszczony obsługi procesów przez planistę krótkoterminowego

Źródło: opracowanie własne

Załączony powyżej rysunek przedstawia uproszczony model obsługi procesów<sup>37</sup>, zawierający symulację sytuacji, w której proces B rozpoczyna oczekiwanie na rezultat operacji wejścia/wyjścia. Po wywołaniu procedury odczytu i stwierdzeniu braku danych w

<sup>37</sup> Dla większej jasności schematu zastosowano pewne uproszczenia. Model nie zawiera procesu IDLE (kończącego swoje wykonywanie każdorazowo przerwaniem BRK). Również w przypadku przerwania IRQ zostało oznaczone tylko to wystąpienie przerwania, które sygnalizuje pojawienie się rezultatu operacji wejścia/wyjścia., wyróżnione natomiast jako niezależne jest przerwanie BRK, będące de facto podtypem przerwania IRQ.

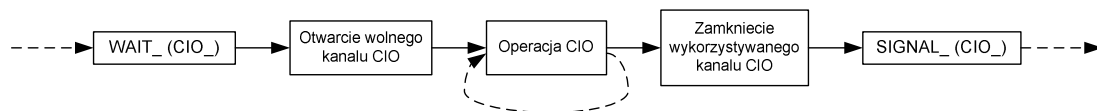


buforze We/Wy za pomocą programowego przerwania BRK wzywany jest kod planisty. Planista zmienia stan procesu na „*waiting for i/o*” i oddaje zasoby następnemu w kolejce procesowi. Zasoby będą od tej pory dzielone pomiędzy procesy A i C, aż do momentu pojawienia się danych – procedura obsługi odpowiedniego przerwania IRQ zapisuje odebrane bajty do bufora, po czym identyfikuje ich odbiorcę. Następnie wywołanie planisty przydzieli mu zasoby poza normalną kolejnością obsługi procesów oczekujących.

#### 5.4.2. Pamięci masowe i urządzenia szeregowo

W obecnej fazie zaawansowania prac nad systemem obsługa urządzeń zewnętrznych innych niż monitor i klawiatura, których obsługą zajmuje się handler TTY, realizowana jest z wykorzystaniem procedur Atari DOS, oraz semaforów binarnych. Koncepcja semaforów opisana jest dokładnie w punkcie 3.6 niniejszego rozdziału.

W okresie przejściowym procedura korzystania z urządzeń zewnętrznych z wykorzystaniem CIO odbywać się powinna według poniższej procedury:



Rys. 5.20. Procedura korzystania z CIO z wykorzystaniem semaforów

Źródło: opracowanie własne

Procedury CIO Atari OS nie są świadome pracy w wieloprotocowym środowisku, dlatego na czas ich wykorzystywania procedura pośrednicząca w wywołaniu JCIOMAIN blokuje przełączanie procesów bitem 7 RUNLEVEL\_.

#### 5.4.3. Urządzenia sieciowe

Koncepcja sieci dla XEUX zakłada wykorzystanie interfejsu *Game Link II*, w możliwych układach od 2 do 8 komputerów. Praca w założeniu ma odbywać się na zasadzie klient-serwer, gdzie serwerem byłby komputer o potencjalnie największych możliwościach (duża ilość pamięci RAM, dysk twardy), natomiast terminale mogłyby stanowić dowolne ośmiobitowe komputery Atari, w tym również najuboższe modele jak Atari 400 czy Atari 600XL, wyposażone tylko w minimalną ilość pamięci.

Terminale nie potrzebują żadnego dodatkowego oprogramowania, całość procesu uruchamiania odbywa się poprzez sieć – wbudowane w system operacyjny Atari OS mechanizmy powodują, iż włączany do prądu komputer próbuje rozpocząć proces inicjalizacji z pierwszego napędu dyskietek. Odpowieni proces na serwerze ma za zadanie prowadzić nasłuch na SIO – jeżeli pojawi się wezwanie od nowowłączonego terminala, proces odpowiada jako napęd pierwszy, serwując nowoprzybyłemu kod programu terminala wzbogacony o odpowiednie nagłówki boot. Po uruchomieniu programu terminala dalsza łączność odbywa się z wykorzystaniem zaszytego w kodzie programu terminala handlera urządzenia sieciowego.

Implementacja tego rozwiązania wymaga uruchamiania komputera-serwera bądź to ze zmodyfikowanym ROM z systemem operacyjnym XEUX, bądź z cartridge'a z systemem XEUX, tak, aby nie było konieczności inicjacji serwera z napędu dyskietek. Ma to na celu uniknięcie potencjalnego konfliktu pomiędzy fizycznym napędem o numerze 1, a procesem BOOTSIOD\_ czuwającym na serwerze.

## **5.5. KOMUNIKACJA MIĘDZYPROCESOWA**

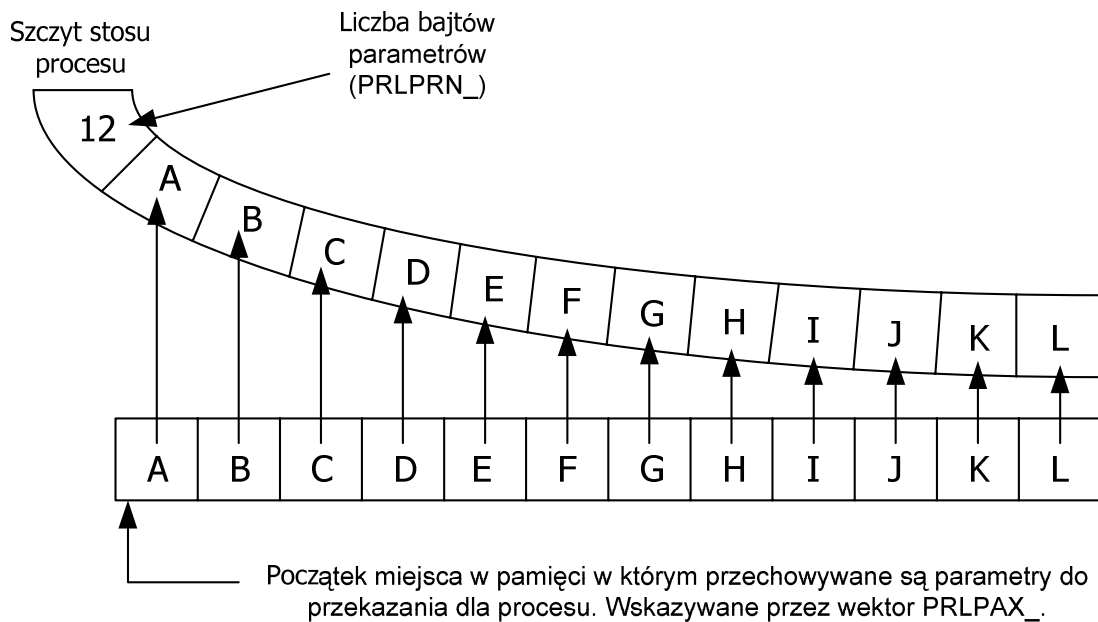
### **5.5.1. Przekazywanie parametrów do procesu**

Przekazywanie parametrów do wywoływanego procesu, zarówno z wykorzystaniem procedury EXEC\_ jak i FORK\_, odbywa się w identyczny sposób. Przed wywołaniem procedury wywołania należy wpisać do rejestru PRLPRN\_ ilość bajtów przekazywanych procesowi potomnemu. Jeżeli wpisana wartość wynosi 0, można pominąć wypełnianie dwubajtowego rejestru PRLPAX\_. Jeżeli natomiast ilość bajtów parametru jest różna od zera, należy do niego wpisać adres początku miejsca w pamięci zawierającego przekazywane wartości.

W końcowej fazie pracy planisty długoterminowego, przy przygotowywaniu procesu do uruchomienia, zostaną one przekazane na stos lokalny procesu potomnego, w takiej kolejności, aby umożliwić ściąganie poszczególnych parametrów ze stosu w porządku rosnącym. Szczytowym elementem umieszczanym na stosie jest liczba parametrów, pozwalająca procesowi potomnemu na sprawdzenie, ile bajtów parametrów zostało mu przekazane.

Należy pamiętać o sprzętowym ograniczeniu wielkości stosu w komputerach opartych o procesor 6502 – wynoszącym 256 bajtów – i nie przekazywać parametrów w ilości większej niż 248 – zachowując pozostałe miejsce na flagi procesora, rejestry oraz licznik programu.

Przykładowa organizacja przekazywanych danych dla 12 bajtów parametrów zilustrowana jest poniżej:



Rys. 5.21. Organizacja przekazywanych parametrów na stosie procesu potomnego

Źródło: opracowanie własne

### 5.5.2. Sygnały

W aktualnej wersji systemu XEUX zaimplementowany jest podzbiór standardowego zestawu sygnałów znanego z systemów UNIX:

Numer	Nazwa skrócona	Pełna nazwa
1	SIGHUP	Terminal line hangup
9	SIGKILL	Kill process
15	SIGTERM	Software termination signal
17	SIGSTOP	Stop (cannot be caught or ignored)
19	SIGCONT	Continue after stop
23	SIGIO	I/O possible
30	SIGUSR1	User defined signal 1
31	SIGUSR2	User defined signal 2

Tabela 5.4. Podzbiór standardowych sygnałów wybrany do implementacji w XEUX

Źródło: man signal(3), *FreeBSD Library Functions Manual*, April 19, 1994,

Trzy z tych sygnałów obsługiwane są zawsze przez procedury systemu operacyjnego i modyfikują bezpośrednio stan procesów. Wysłanie do procesu sygnału SIGSTOP powoduje jego bezwarunkowe zatrzymanie do czasu otrzymania drugiego z opisywanych sygnałów, SIGCONT. Trzecim jest sygnał o wyższym niż pozostałe priorytecie, SIGKILL, który powoduje terminację procesu. Wymienionych sygnałów procesy nie mogą przechwycić ani wyłączyć ich obsługi.

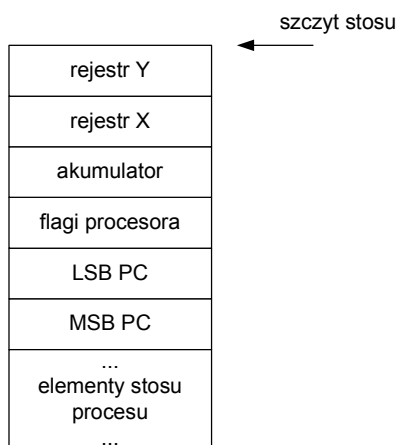
Obsługa pozostałych pięciu sygnałów jest domyślnie wyłączona. Procesom użytkownika udostępnione są mechanizmy pozwalające na zarejestrowanie własnych procedur ich obsługi. Służy do tego procedura SIGIVEC\_ (z ang.: *SIG*nal: *Install VEC*tor). Jako parametry wejściowe należy podać jej w rejestrze Y numer sygnału, dla którego procedurę obsługi chcemy zarejestrować, w rejestrach AX natomiast adres procedury obsługi (odpowiednio LSB adresu do akumulatora i MSB do rejestru indeksowego). Przykładowy kod programu instalującego własną procedurę obsługi sygnału SIGHUP może wyglądać tak:

```

START      LDY #$01          ; numer sygnału
            LDA #<SIGHUP     ; LSB procedury SIGHUP
            LDX #>SIGHUP     ; MSB procedury SIGHUP
            JSR SIGIVEC_     ; instalacja wektora
            ...dalszy kod programu...
SIGHUP     ...kod obsługi sygnału...
            JMP SIGRET_

```

System przekazując obsługę sygnału procedurze użytkownika zachowuje na stosie adres powrotu do programu głównego, flagi procesora oraz rejestry w taki sam sposób, w jaki dzieje się to w momencie przekazywania kontroli do procedury przerwania:



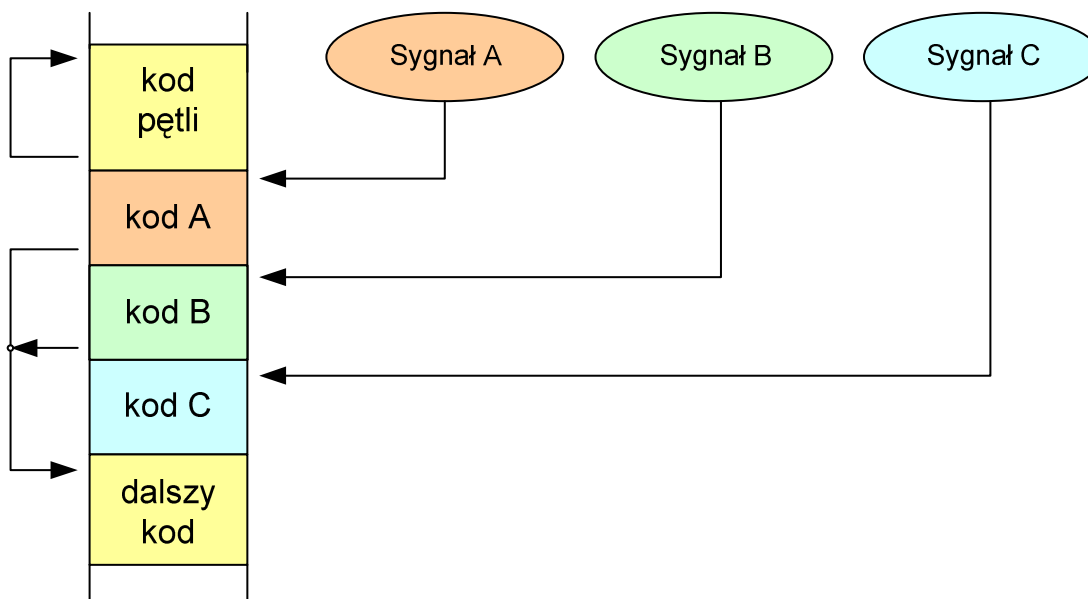
Rys. 5.22. Struktura stosu w momencie przekazania kontroli procedurze obsługi sygnału

Źródło: opracowanie własne

Po zakończeniu pracy procedura powinna powrócić do programu procesu za pomocą skoku bezwarunkowego JMP pod adres procedury systemowej SIGRET\_:

```
; SIGnal: REturn
SIGRET_  PLA  ; ze stosu wartość Y
          TAY  ; do rejestru Y
          PLA  ; wartość X
          TAX  ; do rejestru X
          PLA  ; wartość A do akumulatora
          RTI  ; powrót pod adres PC wraz z
              ; odtworzeniem flag procesora
```

Możliwe jest również wykorzystanie sygnałów jako pewnej formy skoku warunkowego, z którego nie zakładany jest powrót do programu głównego w miejscu, w którym wystąpił dany sygnał:



Rys. 5.23. Wykorzystanie mechanizmu sygnałów do realizacji skoków warunkowych

Źródło: opracowanie własne

W zaprezentowanym przykładzie program krąży w pętli, z której wyjściem jest wystąpienie jednego z kilku obsługiwanych sygnałów. Po jego wystąpieniu wykonywany jest kod specyficzny dla jego obsługi, a następnie skok do dalszej części kodu. Jednym z zastosowań może być sprawdzenie czy nadchodzą informacje z więcej niż jednego urządzenia zewnętrznego lub z urządzenia zewnętrznego oraz od innego programu pracującego współbieżnie.

Realizując taki wariant programista musi pamiętać, aby przed przekazaniem kontroli do dalszej części programu usunąć ze stosu dodatkowe 6 bajtów – PC, flagi oraz rejestry – odłożone w momencie przekazywania obsługi sygnału.

Wysłanie sygnału do procesu odbywa się poprzez wywołanie systemowej funkcji KILL\_ z numerem procesu (w postaci indeksu do strony bazowej) w rejestrze indeksowym X, a numerem sygnału w akumulatorze:

```
SIGTERM    = 15
;
                LDA #SIGTERM
                LDX PROC
                JSR KILL_
                BCC ?OK
?BLAD        ...procedura obsługi błędu...
?OK          ...dalszy kod...
```

Tak, jak dla wszystkich funkcji systemowych i w tym przypadku ustawiona flaga C po powrocie z procedury oznacza wystąpienie błędu, którego kod przekazywany jest w rejestrze indeksowym X.

## 6. PRZYKŁADOWE APLIKACJE

Rozdział ten opisuje przykładowe aplikacje, jakie powstały podczas tworzenia systemu. Są to aplikacje o zastosowaniu czysto użytkowym, tworzone w uproszczonej formie na bieżąco celem testowania implementowanych funkcji. W większości przypadków nie są to aplikacje efektywne, pamiętać jednakże należy, iż są to narzędzia i funkcje, do których zostały stworzone, realizują możliwie najefektywniejszą drogą. Wszystkie wyposażone zostały w interfejs użytkownika sprawdzający podawane parametry i wyświetlający zwięzłą pomoc w koniecznych przypadkach.

### 6.1. APLIKACJE SYSTEMOWE

#### 6.1.1. shell

Powłoka systemowa, zwana potocznie shell, jest interfejsem użytkownika systemu. W przypadku XEUX rolę tę pełni *barszcz*<sup>38</sup>. Jest to prosty shell, rozpoznający zbiór kilku komend, dający również możliwość wczytywania do systemu i uruchamiania nowych procesów. Zbiór komend dla wersji 0.15 zawiera poniższa tabela:

Komenda	Parametry	Funkcja
ver	Brak	Wyświetla aktualną wersję powłoki.
uname	Brak	Wyświetla aktualną wersję systemu, odczytywaną ze stałej systemowej.
rotor	on / off	Włącza lub wyłącza obrazujący stan wykorzystania procesora wiatraczek w lewym górnym rogu.
exit	Opcjonalnie liczba w HEX	Kończy pracę powłoki. Jeżeli jako pierwszy parametr została podana poprawna liczba w HEX, zostanie ona zwrócona jako rezultat do ewentualnego procesu-rodzica.
clear	Brak	Czyści ekran.
?	Brak	Wyświetla krótką pomoc wraz z listą komend.

Tabela 6.1. Lista komend dla *barszcz 0.15*

Źródło: opracowanie własne

---

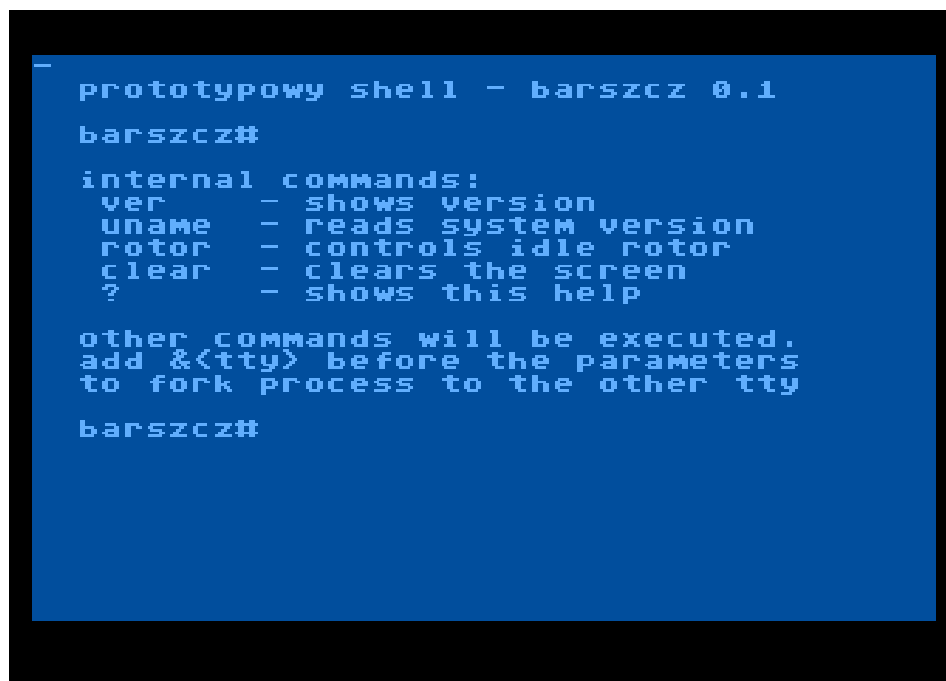
<sup>38</sup> Nazwa nawiązuje do znanej z systemów UNIX powłoki *bash* (skrót od: *Bourne-Again Shell*).

Jeżeli wprowadzona komenda nie zostanie rozpoznana, powłoka uzna ją za komendę zewnętrzną i będzie starać się wczytać z dysku i uruchomić proces o nazwie takiej, jak obcięty do 8 znaków pierwszy człon wprowadzonej linii (ograniczony znakiem spacji), uzupełniony o rozszerzenie formatu .XX. Wszystkie występujące po nazwie elementy, odseparowane od nazwy spacją, zostaną przekazane jako parametry. Proces będzie uruchamiany przez wywołanie systemowe EXEC\_, działanie powłoki zostanie więc wstrzymane do czasu zakończenia działania nowego programu. Możliwe jest wywołanie nowego programu bez zatrzymywania powłoki, należy w tym celu podać jako pierwszy parametr numer konsoli docelowej dla programu, poprzedzony znakiem '&':

```
barszcz# write &3 1
```

Komenda powyższa spowoduje uruchomienie programu *write* na czwartej (ze względu na numerację TTY rozpoczynającą się od 0) konsoli, z zamiarem wysłania wiadomości na konsolę drugą. Numer konsoli wraz ze znakiem '&' i spacją nie liczą się wtedy jako parametr i nie są przekazywane procesowi potomnemu.

Wprowadzanie komend może następować zarówno z użyciem małych, jak i dużych liter (a także dowolnych ich kombinacji). Dostępna jest również jednoliniowa historia komend, którą wywołuje się przez wciśnięcie klawisza strzałki w górę.



```
prototypowy shell - barszcz 0.1
barszcz#
internal commands:
ver      - shows version
uname    - reads system version
rotor    - controls idle rotor
clear    - clears the screen
?        - shows this help

other commands will be executed.
add <tty> before the parameters
to fork process to the other tty
barszcz#
```

Rys. 6.1. Powłoka systemowa

Źródło: opracowanie własne



### 6.1.2. ps

Nazwa PS pochodzi od pierwszych liter angielskich słów *Process Status*. Program jest odpowiednikiem narzędzia o tej samej nazwie znanego z systemów UNIX. Wywoływany z poziomu shella bez podanych parametrów wyświetla listę aktywnych (tj. będących w stanie innym niż \$00 – „*nonexistent*”) procesów w systemie:

```
prototypowy shell - barszcz 0.1
barszcz# ps ?
usage: ps [k]

barszcz# ps

  #  PID  NAME  STATE  LEN  BANK  TTY  flg
--  --  --  --  --  --  --  --
00  00  IDLE  READY  --  --  --  --
01  01  BARSZC  WAIT  05  00  73  01  --
02  02  BARSZC  WAIT  05  00  60  02  --
03  03  BARSZC  WAIT  05  00  67  00  --
04  07  PROC0  STOP  03  00  63  03  --
05  0C  WRITE  WAIT  04  00  5E  01  --
06  0E  PROC3  WAIT  03  00  5A  00  --
07  10  PS     RUN   05  00  54  02  --

barszcz#
```

Rys. 6.2. Program PS w działaniu

Źródło: opracowanie własne

Każdy wiersz wyniku działania programu odpowiada jednemu procesowi i zawiera następujące informacje:

- # - numer pozycji w tablicy procesów PR.TABLE\_
- PID - numer identyfikacyjny procesu (z zakresu 0-255).
- NAME - sześć pierwszych liter nazwy procesu
- STATE - aktualny stan procesu
- LEN - ilość stron pamięci zajmowanych przez proces
- BANK - numer banku w którym rezyduje proces
- PAG - strona pamięci z początkiem kodu procesu
- TTY - numer konsoli przypisanej do procesu

- flg            - obecność jednej z dwóch flag statusu:
  - I            - na proces oczekuje wynik operacji wejścia/wyjścia
  - S            - na proces oczekuje sygnał

Wywołanie PS z parametrem 'k' powoduje uruchomienie programu w trybie pracy cyklicznej, wyświetlając aktualne informacje po każdym naciśnięciu klawisza. Wyjście z tego trybu następuje po naciśnięciu klawisza 'q'. Uruchomienie programu z dowolnym innym parametrem powoduje wyświetlenie informacji o akceptowalnych sposobach wywołania programu.

### 6.1.3. kill

KILL jest programem realizującym te same funkcje co jego imiennik w systemach UNIX – wysyła sygnał do wskazanego procesu.

```

barszcz# ps
  #   PID   STATE   LEN   BANK   TTY   flg
  +---+---+---+---+---+---+---+---+
  00  00  IDLE   READY   --   --   --   --
  01  01  BARSZC WAIT    05  00  79  01  - -
  02  02  BARSZC WAIT    05  00  73  02  - -
  03  03  BARSZC WAIT    05  00  60  00  - -
  04  05  PS      RUN     05  00  67  01  - -

barszcz# kill -9 02
barszcz# kill -17 03
barszcz# ps
  #   PID   STATE   LEN   BANK   TTY   flg
  +---+---+---+---+---+---+---+---+
  00  00  IDLE   READY   --   --   --   --
  01  01  BARSZC WAIT    05  00  79  01  - -
  02  08  PS      RUN     05  00  73  01  - -
  03  03  BARSZC STOP     05  00  60  00  - -

barszcz#
  
```

Rys. 6.3. Program KILL w działaniu

Źródło: opracowanie własne

Uruchamiany z linii komend, jako parametr przyjmuje numer sygnału do wysłania poprzedzony znakiem minusa, przy czym numer ten, dla zachowania zgodności z innymi platformami, jest numerem podanym w formie liczby dziesiętnej, jedno- lub dwucyfrowej.

Drugim parametrem jest identyfikator procesu, PID, podawany w formie dwucyfrowej liczby heksadecymalnej. Za przyjęciem tej formy przemawia fakt, iż w ten sam sposób wyświetlany jest on przez *ps* oraz *monitor systemowy*, gdzie wybór sposobu reprezentacji numeru podporządkowany był konieczności oszczędzania miejsca na ekranie. Wywołanie KILL bez parametrów lub z parametrami w niewłaściwej formie spowoduje wyświetlenie informacji o sposobie użycia narzędzia.

Na załączonej kopii ekranu (Rys. 6.3) zaobserwować możemy efekt działania aplikacji wysyłającej dwa sygnały: 9 (SIGKILL) – proces o identyfikatorze 02 został zlikwidowany przez system, oraz 17 (SIGSTOP) – proces 03 został zawieszony do czasu odebrania sygnału 19 (SIGCONT).

#### 6.1.4. Monitor systemowy

Monitor systemowy jest aplikacją wbudowaną w system operacyjny, nie działającą w trybie wieloużytkownikowym. Wywołanie jego odbywa się po naciśnięciu klawisza HELP i powoduje zatrzymanie pracy wieloprocessowej w najbliższym możliwym momencie po obsłużeniu bieżącego procesu, oczekujących zdarzeń wejścia/wyjścia oraz sygnałów.



Rys. 6.4. Monitor systemowy

Źródło: opracowanie własne

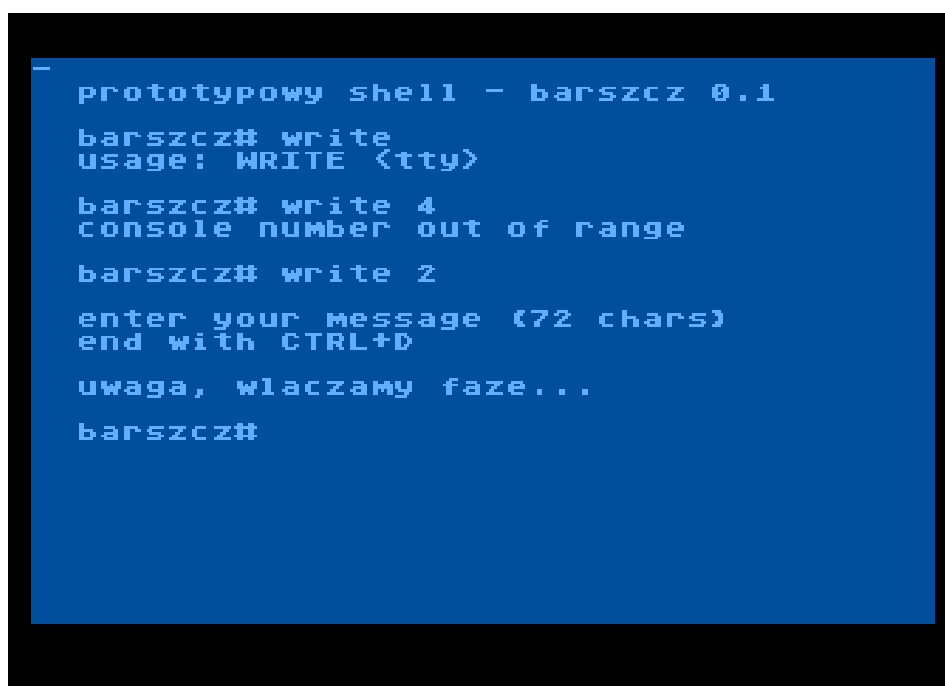
Sposób pracy monitora wymuszony jest zestawem funkcji, jakie musi spełniać. Monitor systemowy pozwala bowiem na przeglądanie nie tylko informacji o stronie bazowej procesów, ale również na zagłębienie do pamięci zaalokowanej dla nich – co za tym idzie nie może być procesem działającym w banku pamięci XMS.

Przełączanie numeru oglądanego procesu odbywa się z użyciem klawiszy „<” i „>”. Wybór okna podglądu pomiędzy wglądownicą pamięci a zawartością stosu procesu dokonywany jest klawiszem TAB. Do nawigacji w ramach okien podglądu służą strzałki kursora, natomiast adres skoku do żadanego obszaru pamięci wprowadzić można po wciśnięciu kombinacji SHIFT+G.

## 6.2. PROGRAMY UŻYTKOWNIKA

### 6.2.1. write

WRITE to bardzo prosta aplikacja, również rodem z systemów UNIX. Z jej pomocą możliwe jest nadanie wiadomości skierowanej bezpośrednio na inną konsolę systemu.



```
prototypowy shell - barszcz 0.1
barszcz# write
usage: WRITE <tty>
barszcz# write 4
console number out of range
barszcz# write 2
enter your message (72 chars)
end with CTRL+D
uwaga, włączamy faze...
barszcz#
```

Rys. 6.5. Program WRITE – wprowadzanie wiadomości

Źródło: opracowanie własne

Wywołanie procedury bez parametrów spowoduje wyświetlenie informacji o konieczności podania jako parametru numeru konsoli odbiorcy wiadomości. W przypadku podania numeru nieistniejącej w systemie konsoli, program poinformuje o tym wyświetlając stosowny komunikat o błędzie.

Wiadomość może mieć długość do 72 znaków (dwie linie przy standardowych marginesach) i wprowadzana jest z klawiatury po wywołaniu programu. Dopuszczalne znaki to również znaki przeniesienia do nowej linii. Wprowadzanie wiadomości należy zakończyć wciśnięciem kombinacji klawiszy CTRL+D. Wciśnięcie CTRL+C przerywa program. Po zakończeniu wprowadzania wiadomość jest niezwłocznie wysyłana na ekran odbiorcy:



```
prototypowy shell - barszcz 0.1
barszcz# ps
  #  PID  NAME      STATE   LEN  BANK  TTY  flg
--+---+---+---+---+---+---+---+---+---+
00  00  IDLE      READY   --   --   --   --
01  01  BARSZC  WAIT    05  00  79  01  --
02  02  BARSZC  WAIT    05  00  73  02  --
03  03  BARSZC  WAIT    05  00  6D  00  --
04  06  WRITE    WAIT    04  00  68  01  --
05  07  PS       RUN     05  00  62  02  --

barszcz#
Message from TTY 1:
uwaga, włączamy faze...
```

Rys. 6.6. Program WRITE – wiadomość odebrana

Źródło: opracowanie własne

Sam program jest bardzo prosty, większość kodu to procedury odpowiedzialne za interfejs użytkownika, wyświetlające komunikaty, pobierające wprowadzane z konsoli elementy do bufora wiadomości, rozpoznające klawisze kontrolne. Właściwe wysyłanie wiadomości realizowane jest z wykorzystaniem systemowej funkcji WRITE\_, której obsługa w dowolnym programie jest szybka i nieskomplikowana.

Poniżej, jako demonstracja łatwości tworzenia aplikacji korzystających z gotowych funkcji systemowych, załączony jest fragment kodu programu WRITE, odpowiedzialny za wysłanie wiadomości. Wcześniejsza jego partia zadbała o to, aby przed wywołaniem w obszarze danych oznaczonym etykietą MESSAGE znalazła się wiadomość, natomiast w komórce opisanej MSGLEN jej długość.

```
    ...
    PLA          ; ze stosu numer TTY adresata
    TAY          ; przepisujemy do rejestru Y
    LDA #<MSGLEN
    LDX #>MSGLEN ; LSB/MSB początku bufora
                  ; wiadomości do rej. AX
    JSR WRITE_
    ...
MSGLEN .BY $00      ; dlugosc wiadomosci
MESSAGE .DC 73 00   ; zdefiniowany bufor na
                   ; tresc wiadomosci
```

Numer konsoli, na której ma zostać wyświetlona wiadomość, przekazywany jest w rejestrze indeksowym Y, adres bufora natomiast w rejestrach A oraz X. Pierwszy element powinien zawierać długość wiadomości.

## 7. PODSUMOWANIE

Celem pracy było stworzenie od zera wieloprocesowego systemu operacyjnego dla ośmiobitowych komputerów Atari XL/XE, w celu potwierdzenia tezy, iż stworzenie takiego systemu dla niezmodyfikowanego komputera tej serii jest możliwe. Produkt miał umożliwiać jednoczesną pracę w systemie wielu programów użytkownika, udostępniając przyszłym autorom oprogramowania gotowy interfejs programistyczny pozwalający tworzyć pełnosprawne aplikacje komunikujące się zarówno ze światem zewnętrznym za pośrednictwem urządzeń peryferyjnych, jak i pomiędzy samymi sobą za pomocą mechanizmów oferowanych przez system.

Ze względu na konieczność możliwie jak największej optymalizacji kodu oraz ścisły kontakt ze sprzętem, całość systemu musiała być pisana w assemblerze. Jako, że nigdy żaden projekt tego typu nie doczekał się realizacji, nie było bazy doświadczeń, z której można by skorzystać podczas realizacji zamierzeń, ani tym bardziej gotowego środowiska pracy w postaci chociażby bibliotek gotowych procedur.

Mimo tego, iż zadanie okazało się być bardziej skomplikowane, a przede wszystkim o wiele bardziej czasochłonne niż można się tego było spodziewać, projekt został zrealizowany. Powstały produkt jest spójnym, działającym systemem operacyjnym, a jego główne cechy to:

- współbieżna praca do 32 procesów w systemie,
- 4 niezależne, wirtualne konsole,
- zarządzanie procesami (w tym wywłaszczanie) realizowane przez planistę krótkoterminowego, uruchamianego cyklicznie w takt przerwania synchronizacji pionowej oraz w razie potrzeby w efekcie wystąpienia przerwania programowego,
- zunifikowany podsystem obsługi urządzeń, samoczynnie zwalniający zasoby procesora w momencie oczekiwania procesu na rezultat operacji wejścia/wyjścia,
- elastyczne zarządzanie dostępną pamięcią, zapewniające dopasowany do potrzeb procesu liniowy przydział pamięci na segment kodu i – dostępnych za pomocą naturalnych trybów adresowani – danych procesu,

- wirtualne tryby adresowania, służące do przechowywania większych segmentów danych,
- procesy oczekujące na rezultat operacji wejścia/wyjścia – w momencie jej zakończenia – przywracane do działania poza kolejnością,
- kod wprowadzanego procesu relokowany w przydzielone miejsce pamięci, a dołączone do dystrybucji programy użytkowe umożliwiają łatwe tworzenie własnych relokowalnych plików wykonywalnych gotowych do pracy w nowym środowisku,
- synchronizacja dostępu procesów do krytycznych zasobów realizowana jest z wykorzystaniem semaforów,
- obsługa sygnałów, których procedury programy użytkownika mogą samodzielnie rejestrować w strukturach systemu,
- możliwość przekazywania parametrów procesom potomnym uruchamianym z wewnątrz procesu rodzica – podobnie też rodzic może odebrać rezultat pracy procesu potomnego po jego terminacji,

Wszystkie wymienione funkcje, poza wirtualnymi trybami adresowania, dostępne są również w przypadku uruchamiania systemu na niezmodyfikowanym, fabrycznym Atari 65XE. Dostęp do wirtualnych trybów adresowania wymaga przynajmniej 64kB dodatkowej pamięci XMS, dostępnej już w Atari 130XE.

Źródłem niezbędnej dla realizacji celów wiedzy były zarówno aktualne publikacje dotyczące budowy i mechanizmów występujących w nowoczesnych systemach operacyjnych, jak i liczące sobie niekiedy ponad 20 lat materiały opisujące architekturę sprzętową komputera. Podczas pracy wykorzystywano nie tylko klasyczne pozycje drukowane, czy dostępne w sieci Internet, ale również artykuły publikowane w ubiegłowiecznych magazynach dyskowych.

Istotne doświadczenia zdobyte podczas realizacji projektu to nie tylko zdolność programowania w języku niskiego poziomu (realizowany temat był pierwszym pisanym w assemblerze projektem autora), ale przede wszystkim doskonała nauka dyscypliny – nad skomplikowanym programem w tym języku nie można pracować skokowo, konieczna jest regularna praca, w większości składająca się nie z pisania samego kodu, ale z jego testowania we wszystkich możliwych przypadkach, śledzenia błędów czy rozpisywania algorytmów dla



przyszłych procedur. Bez skrupulatnej dokumentacji, powrót do kodu źródłowego po kilku tygodniach powoduje konieczność odtworzenia całości algorytmu, aby odnaleźć istotny fragment, co skutecznie zniechęca do niedbałości w tym obszarze.

## 7.1. ROZWÓJ PROJEKTU

Rozwojową wersją systemu może być system operacyjny dedykowany dla komputerów Atari XL/XE wyposażonych w jedno z dostępnych rozszerzeń o procesor 65816, umożliwiających podłączenie liniowej pamięci w przestrzeni powyżej podstawowych 64kB. Zastosowanie naturalnego trybu 65816 pozwoliłoby znacznie uprościć niektóre elementy systemu:

- Stałoby się możliwe wyeliminowanie najbardziej obecnie czasochłonnego fragmentu przełączania kontekstu, mianowicie przepisywania strony zerowej. Procesor 65816 posiada rejestr D, dzięki któremu jako strona zerowa (zwana od tego momentu stroną bazową) może działać dowolna wskazywana przezeń strona pamięci w obszarze pierwszych 64kB RAM. Efektywnie zyskujemy więc 256 stron bazowych przełączanych kombinacją dwóch rozkazów.
- Wykorzystanie nowych rozkazów pozwoli na dalszą optymalizację kodu (m.in. bezpośrednie transfery pomiędzy rejestrami indeksowymi, przechowywanie ich wartości na stosie bez pośrednictwa akumulatora, odgałęzienia bezwarunkowe, kopiowanie bloków pamięci, etc.).
- Zastosowanie liniowej pamięci oraz nowego rejestru DBR (od ang.: *Data Bank Register*) pozwalającego na wskazanie 64kB banku pamięci, który ma być używany dla podstawowego zestawu rozkazów o 16-bitowych argumentach daje możliwość zwiększenia segmentu kodu i danych szybkich procesu z 15 do 64kB.
- 24-bitowe tryby adresowania zlikwidują konieczność używania przez procedury systemu operacyjnego bufora do wymiany danych pomiędzy procesami (dotychczas taka konieczność wynikała z faktu, iż procesy mogą znajdować się w różnych bankach pamięci dostępnych w tej samej przestrzeni adresowej).

Wpływ na zwiększenie szybkości działania miałby też fakt, iż oba obecnie dostępne rozwiązania zapewniające dostęp do pamięci liniowej powodują również przyspieszenie procesora podczas korzystania z tego dodatkowego obszaru.

## 8. ZAŁĄCZNIKI

### 8.1. INSTRUKCJA UŻYTKOWNIKA

#### 8.1.1. Organizacja płyty CD

Na dołączonej do niniejszej pracy płycie CD, w katalogu IMG, znajdują się następujące pliki:

- |            |   |
|------------|---|
| - XEUX.ATR | - obraz dyskietki SS/ED w formacie Atari DOS 2.5 zawierającej binarne pliki z kodem systemu oraz programami narzędziowymi demonstrującymi możliwości systemu                      |
| - SRC.ATR  | - obraz dyskietki SS/ED w formacie Atari DOS 2.5 zawierającej pliki źródłowe w formacie MAE (dla źródeł w assemblerze) i TBS (w przypadku narzędzi napisanych w Turbo Basicu XL). |
| - UTIL.ATR | - obraz dyskietki SS/ED w formacie Atari DOS 2.5 zawierającej programy użytkowe (pliki wykonywalne Atari DOS) wykorzystywane podczas tworzenia systemu                            |

Pliki wyżej wymienione mogą posłużyć do utworzenia fizycznej dyskietki, jako źródło dla oprogramowania emulującego stację dysków dla komputera Atari podłączonego za pomocą kabla SIO2PC (np. APE), lub też bezpośrednio dla uruchamianego na komputerze PC emulatora Atari (np. Atari800WinPLus).

Katalog EMU zawiera plik instalatora emulatora Atari800WinPLus, oraz plik XF25.ZIP zawierający pakiet dystrybucyjny emulatora PC Xformer 2.5<sup>39</sup>.

W katalogu DOC umieszczony jest niniejszy plik w formacie Microsoft Word.

---

<sup>39</sup> Obrazy pamięci ROM Atari XL/XE są chronione prawem autorskim. Autor emulatora PC Xformer 2.5 jako jedyny uzyskał od właściciela praw do wzmiankowanych obrazów pozwolenie na dystrybucję ich wraz ze swoim emulatorem – dystrybuowanym na zasadach freeware.

### 8.1.2. Instalacja emulatora Atari

Aby zainstalować emulator, należy z poziomu systemu Windows uruchomić plik instalatora (Atari800Win PLus 4.0.exe), a następnie postępować zgodnie z zaleceniami na ekranie. Przed pierwszym uruchomieniem należy rozpakować zawartość pliku XF25.ZIP. Po uruchomieniu emulatora powinniśmy wskazać miejsce do którego rozpakowano plik ZIP jako lokalizację plików ATARIBAS.ROM, ATARIOSB.ROM oraz ATARIXL.ROM.

Funkcjonowanie załączonych plików przetestowano korzystając z następujących ustawień emulatora:

```
Atari -> Machine Type -> XL/XE
Atari -> Memory Size -> 128kB
Atari -> Video System -> PAL
Atari -> Options -> Disable BASIC
```

Aby podłączyć obrazy dyskielek, należy wykorzystać skróty klawiszowe ALT plus numer napędu, który chcemy symulować. Napęd, z którego następuje rozruch systemu (boot), to napęd pierwszy.

### 8.1.3. Praca z XEUX

Bezpośrednio po inicjalizacji systemu na dwóch<sup>40</sup> pierwszych konsolach wirtualnych uruchamiana jest powłoka systemu dająca użytkownikowi dostęp do interfejsu komend, opisanego w rozdziale szóstym. Dostępne są cztery konsole wirtualne, pomiędzy którymi przełączanie wykonuje się wciskając kombinację klawiszy SHIFT, CONTROL oraz numeru konsoli z zakresu od 1 do 4 (przy czym pamiętać należy że w systemie mają one numery pomniejszone o 1, toteż podając numer konsoli jako parametr do programów narzędziowych należy zawsze stosować tą regułę).

Wciśnięcie klawisza HELP powoduje wejście do monitora systemowego, natomiast klawisz RESET zakończy pracę systemu i powróci do Atari DOS.

---

<sup>40</sup> Domyślne uruchamianie powłok jedynie na dwóch pierwszych konsolach spowodowane jest chęcią zminimalizowania wykorzystania pamięci na starcie systemu (istotne szczególnie w przypadku komputerów Atari 65XE/800XL). W przypadku potrzeby wykorzystania większej ilości powłok, można je uruchomić z poziomu dowolnej już działającej przez wskazanie docelowej konsoli po znaku &, np.: „barszcz &3” dla TTY 3.

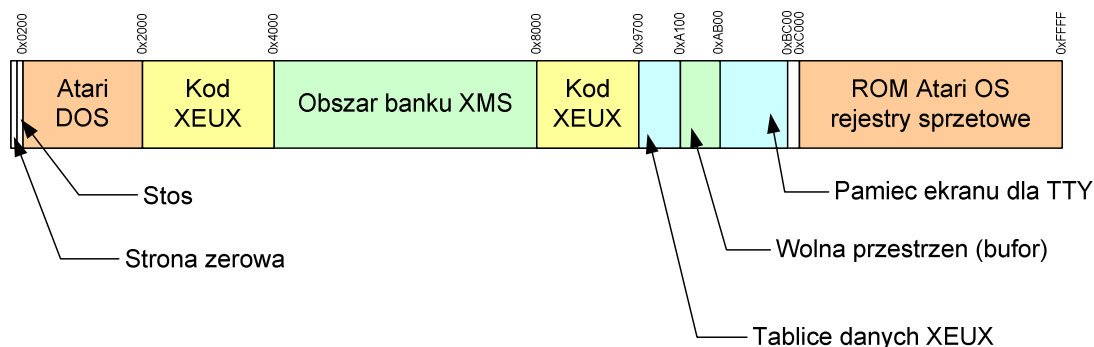
## 8.2. INSTRUKCJA PROGRAMISTY

### 8.2.1. Mapa pamięci

Ze względu na konieczność wspomagania się we wczesnej fazie rozwoju, w jakiej znajduje się XEUX, gotowymi procedurami obsługi filesystemu Atari DOS lub Sparta DOS, oraz procedurami obsługi twardego dysku, projekt wykorzystuje obszary pamięci nie kolidujące z ROM Atari OS oraz przestrzenią cartridge'a Sparta DOS. W przyszłości nie będzie potrzeby wykorzystywania DOS, więc procedury XEUX będą mogły przesunąć się zaraz do krawędzi stosu, możliwe też będzie całkowite odłączenie pamięci ROM Atari, uwalniając kolejne 16kB RAM.

Obecnie procedury systemowe XEUX rozpoczynają się od adresu 0x2000 i zajmują miejsce aż do granicy banku – tj. 0x4000. Następne 16kB pamięci, z wyłączeniem ostatniej strony przeznaczonej na tablicę alokacji, dostępne jest w całości na potrzeby uruchamianych aplikacji. W komputerach 130XE oraz w modelach wyposażonych w dodatkową pamięć XMS w miejsce to podłączane są kolejne banki pamięci o takim samym zastosowaniu.

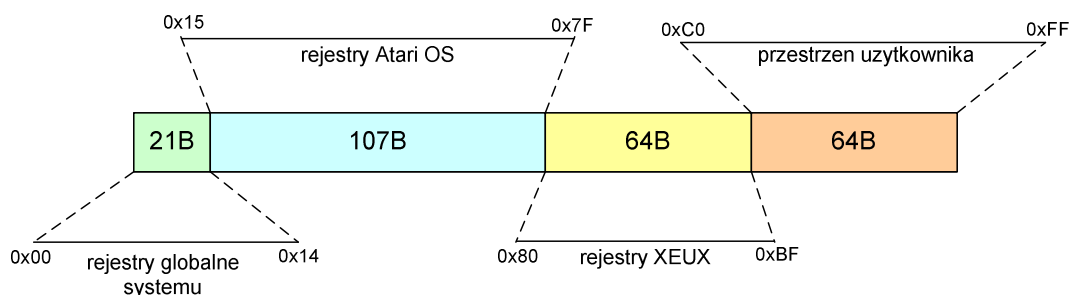
Przestrzeń pomiędzy 0x8000 a 0x96FF to dalsza część kodu systemu XEUX. Następne 2.5kB zajmują tablice danych systemu, takie jak strona bazowa, wektory procedur obsługi sygnału, zmienne oraz stałe systemowe nie wymagające pozostawiania na stronie zerowej. Obszar od 0xA100 do 0xAA99 to pamięć wykorzystywana jako bufor dla operacji wczytywania procesu. Pamięć ekranu dla wirtualnych konsol oraz program dla procesora ANTIC zajmują przylegającą do bufora przestrzeń rozciągającą się od 0xAB00 do 0xBBFF. Górna ćwiartka pamięci RAM przykryta jest przez ROM Atari OS oraz rejestry sprzętowe.



Rys. 8.1. Organizacja przestrzeni adresowej Atari XE pod kontrolą XEUX

Źródło: opracowanie własne

Strona zerowa zasługuje na szczególne wyróżnienie, ze względu na swoje znaczenie dla efektywnego programowania i fakt, iż niektóre tryby adresowania wymagają wykorzystania jej dla przechowywania wektora wskazującego obszary danych. Podzielona jest na cztery fragmenty:



Rys. 8.2. Organizacja strony zerowej

Źródło: opracowanie własne

Przestrzeń 0x00-0x14 zawierają wartości globalne dla systemu, w tym np. liczniki czasu systemowego, bądź zmienne robocze planisty krótkoterminowego. Ten fragment strony zerowej nie podlega przełączaniu w ramach kontekstu procesu.

Zmienne systemowe Atari OS rozlokowane są od adresu 0x15 do 0x7F. Większość z nich nie jest wykorzystywana przez procedury, z których korzysta XEUX, jednakże dla uproszczenia w fazie przejściowej – do czasu całkowitej eliminacji procedur systemowych – przyjmuje się, że przestrzeń ta nie powinna być wykorzystywana przez programistów.

Przeostatnia ćwiartka strony zerowej – adresy od 0x80 do 0xBF – to stałe systemowe i zmienne robocze XEUX. Wykorzystują ją między innymi te funkcje systemowe, które mogą być wywoływane przez procesy, a więc mogą pojawić się w tym samym czasie w kilku instancjach.

Istotne dla programisty komórki z tego zakresu to:

Adres	Etykieta	Funkcja
\$80-\$81	TTY_	Zmienna wskazująca adres początku pamięci dla konsoli wirtualnej przydzielonej procesowi.
\$A9	PRLPRN_	Wskazuje funkcji PRLOAD_ czy (wartość \$00) lub ile (wartość w bajtach) parametrów jest przekazywanych nowotworzonemu procesowi.
\$AA-AB	PRLPAX_	Wskazuje adres początku bufora z którego PRLOAD_ ma pobrać parametry do przekazania tworzonemu procesowi.

\$AC-AD	PRFNAME_	Przed wywołaniem FORK_ lub EXEC_ należy tu wpisać adres miejsca w pamięci w którym umieszczona jest nazwa pliku binarnego do wczytania (łącznie z identyfikatorem urządzenia).
\$AF-\$BF	Etykiety IOCB poprzedzone znakiem X	Komórki wymiany podsystemu XXCIO.

Tabela 8.1. Zmienne systemowe z zakresu 0x80-0xBF przydatne dla programisty

Źródło: opracowanie własne

Ostatni fragment strony zerowej przeznaczony jest całkowicie do dowolnego wykorzystania przez kod procesu.

Trzy z wymienionych części strony zerowej, czyli przestrzeń od 0x15 do 0xFF, jest lokalna dla każdego procesu. Jej kopiowaniem i przechowywaniem zajmuje się kod planisty – stanowi to część procedury przełączania kontekstu.

#### 8.2.2. Dostęp do urządzeń

Dostęp do urządzeń w systemie XEUX zapewniany jest poprzez podsystem XXCIO<sup>41</sup>, będący odpowiednikiem CIO znanego z Atari OS. Zachowany został podstawowy zamysł projektantów systemu, aby jednak mógł on funkcjonować w środowisku wieloprocessowym koncepcja musiała ulec pewnym modyfikacjom, a ze względu na to procedury podsystemu zostały napisane od nowa.

Pozostała idea kanałów oraz bloków kontrolnych IOCB, jednakże mają one teraz znaczenie lokalne dla procesu. Wprowadzono również procedurę wyszukiwania pierwszego wolnego kanału. Ponieważ miejsce w pamięci przechowujące informacje o kanałach jest zmienne, zapisywanie i odczytywanie danych tam umieszczonych odbywa się przez obszar wymiany leżący na stronie zerowej. Poniżej przykładowy kod realizujący otwarcie kanału do odczytu:

```
XICCMD      = $B1
XICBUFA     = $B3
```

<sup>41</sup> Pierwotna nazwa dla podsystemu, XCIO, została zmieniona, aby uniknąć przyszłych nieporozumień – taką samą nazwę posiada rozszerzenie podsystemu CIO jakie na swoje potrzeby wykonuje Sparta DOS X.

```

XICAX1    = $B9
;
          JSR IOCBLKP_      ; poszukiwanie wolnego
                                ; kanalu, zwrot w X
          BCS ?ERR
          LDA #$03          ; OPEN
          STA XICCMD
          LDA #<FNAME       ; nazwa pliku
          STA XICBUFA
          LDA #>FNAME
          STA XICBUFA+1
          LDA #$04          ; READ
          STA XICAX1
          JSR XXCIOPUT_     ; wprowadzenie danych
                                ; do przestrzeni kanalu
                                ; nr X/16
          JSR XJCIOMAIN_    ; wywołanie glownej
                                ; procedury XXCIO.
          ...
?ERR      ...obsługa błedu...

```

### 8.2.3. Asembler 6502

Asembler jest językiem programowania niskiego poziomu, zaliczanym do kategorii języków symbolicznych. Jednemu rozkazowi asemblera odpowiada odpowiedni kod maszynowy dla danego procesora, ułatwiając znacznie pisanie programów – kod jest przejrzysty i o wiele łatwiejszy do analizy, niż ciąg cyfr kodu maszynowego. Kody operacji zastąpione są w asemblerze tzw. mnemonikami, będącymi trzyliterowymi skrótami nazwy operacji w języku angielskim. W celu dodatkowego ułatwienia, znane lokalizacje pamięci bądź wartości numeryczne można zapisać w formacie tekstowych etykiet. Istnieje wiele rodzajów asemblerów dla procesora 6502, różniących się dodatkowymi funkcjami, jak np: asemblacja warunkowa, makrorozkazy, zbiór pseudorozkazów czy etykiety lokalne, jednakże główne elementy, jak zbiór rozkazów czy tryby adresowania pozostają te same, są one bowiem zależne od języka maszynowego dla danego procesora.

Procesor MOS 6502 ma 13 trybów adresowania<sup>42,43,44</sup>.

---

<sup>42</sup> Na podstawie: H. Kruszyński, K. Kulpa, *Mikroprocesor 6502 i jego rodzina*, Warszawa 1989, Wydawnictwo Czasopism i Książek Technicznych NOT-SIGMA

<sup>43</sup> H. Kruszyński, K. Kulpa, *Mikroprocesor 6502 i jego rodzina*, Warszawa 1989, Wydawnictwo Czasopism i Książek Technicznych NOT-SIGMA błędnie wymienia tryb *adresowania pośredniego na stronie zerowej*, dotyczący rozkazów ADC, AND, CMP, EOR, ORA, SBC oraz LDA i STA, jednakże żadna z komend procesora

Z *adresowaniem niejawnym* (ang.: *implicit*) mamy do czynienia wtedy, kiedy rozkaz (jednobajtowy, bez dodatkowego argumentu) zawiera w sobie informację co i gdzie powinno być wykonane. Rozkazami takimi są m.in.: CLC (*CLear Carry* – kasuje flagę Carry procesora), DEX (*DEcrement X register* – zmniejsza o 1 wartość rejestru X) czy RTI (*ReTurn from Interrupt* – rozkaz powrotu z procedury obsługi przerwania).

W przypadku *adresowania akumulatora* (ang.: *accumulator*) argumentem rozkazu jest zawartość akumulatora. Przykładami dla tego trybu adresowania są np.: ROL (ROTate in Left direction – przesuwają bity argumentu o jeden w lewo, z przeniesieniem), LSR (Logical Shift Right – przesunięcie bitowe w prawo, bit 7 zostaje wyzerowany), jednakże mogą one również być zastosowane w stosunku do pamięci. W zależności od typu asemblera fakt adresowania akumulatora zaznaczany jest dodatkowo przez literę A następującą po rozkazie (np. „ASL A”), lub – jak w przypadku MAE – przez całkowity brak argumentu.

Dla *adresowania natychmiastowego* (ang.: *immediate*) jako argument brany jest bajt bezpośrednio następujący po rozkazie, a nie miejsce w pamięci. W tym trybie funkcjonują m.in. rozkazy: LDA (LoaD Accumulator – załadowanie wartości do akumulatora) czy CMP (CoMPare – porównanie akumulatora z wartością argumentu). W kodzie źródłowym argument dla tego trybu poprzedzany jest znakiem #, przykładowo: „LDA #55”.

*Adresowanie bezwzględne* (ang.: *absolute*) polega na pobraniu z bezpośrednio następującego po rozkazie bajtu (LSB) oraz bajtu kolejnego (MSB) adresu argumentu. Przykłady rozkazu to m.in.: JSR (Jump to SubRoutine – skok do podprogramu), znany już ROL, czy INC (INCrement – zwiększa argument o jeden). Przykładowy wygląd w kodzie źródłowym to „STA \$4000” (w kodzie maszynowym reprezentowane jako ciąg liczb HEX: 8D0040).

Kolejnym trybem adresowania jest *adresowanie na stronie zerowej* (ang.: *zero page*). Ze względu na pewność, że w przypadku odwoływania się do strony zerowej bardziej znaczący bajt adresu zawsze będzie równy 0, adres argumentu podawany po komendzie może być jednobajtowy. Zbiór rozkazów jest podzbiorem rozkazów dla trybu adresowania bezwzględnego, z wyłączeniem JMP i JSR. Przykład: „LDA \$01” pobiera do akumulatora zawartość komórki pamięci o adresie 0x0001. W kodzie maszynowym rozkaz ten reprezentowany jest przez dwa bajty: A501.

---

6502 nie używa tego trybu adresowania. Podobnie nie istnieje rozkaz używający *adresowania indeksowego pośredniego* nie będącego trybem strony zerowej.

<sup>44</sup> Źródła podają 13 trybów rozdzielając *Adresowanie indeksowe bezwzględne* na dwa tryby, w zależności od wykorzystywanego rejestru indeksowego. To samo dotyczy *adresowania indeksowego strony zerowej*.



*Adresowanie indeksowe bezwzględne* (ang.: *absolute indexed*) to dwa bliźniacze tryby, zbliżone do adresowania bezwzględnego, jednakże 16-bitowy adres (pobrany z 2 i 3 bajtu rozkazu) jest zmodyfikowany przez dodanie wartości rejestru indeksowego X lub Y (w zależności od tego, którego z dwóch trybów użyto). Tryby te stosowane są często przy odwoływaniu się do tablic danych – w rozkazie podajemy wtedy adres tablicy, wartość zwiększonego o jeden rejestru indeksowego wskazuje natomiast pozycję w tablicy. Przykład zapisu: „LDA \$4000,X”.

*Adresowanie indeksowe strony zerowej* (ang.: *zero page indexed*) jest podobne do omawianego adresowania indeksowego bezwzględnego, jednakże dla tablicy umieszczonej na stronie zerowej – po komendzie podajemy jedynie młodszy bajt adresu. Przykładowo: „LDY \$13,X” lub „LDX \$13,Y”.

Adresy tzw. odgałęzień warunkowych wykorzystują tryb *adresowania względnego* (ang.: *relative*), w którym adres skoku podajemy jako liczbę ze znakiem (w notacji U2, a więc w zakresie -128 do +127), która celem obliczenia wartości bezwzględnej adresu dodawana jest do adresu początku następnego rozkazu. Zaletą tego rozwiązania jest skrócenie kodu i przyspieszenie jego wykonywania. Przykładowym zapisem jest np.: „BCC 7”, lecz praktycznie nigdy nie wykorzystywanym: w praktyce po rozkazie BCC zawsze następuje nazwa etykiety fragmentu kodu, do którego ma nastąpić skok, np.: „BCC SIUP”.

*Adresowanie pośrednie* (ang.: *indirect*) wykorzystuje tylko rozkaz skoku JMP. Dwa bajty podane za komendą JMP wyznaczają adres z pamięci, z którego należy pobrać LSB i MSB adresu pod jaki należy wykonać skok. Notacja: „JMP (\$4000)”. Jeżeli w komórce 0x4000 znajduje się wartość \$45, a w komórce 0x4001 wartość \$20, skok zostanie wykonany pod adres \$2045.

*Adresowanie indeksowe pośrednie*, zwane też *adresowaniem pośrednim preindeksowanym* (ang.: *indexed indirect*) jest zbliżone do adresowania pośredniego, z dwoma różnicami – adres w pamięci, z którego należy pobrać adres dla skoku, jest wyliczany przez dodanie zawartości rejestru indeksowego X do wartości liczbowej następującej po komendzie, dodatkowo miejsce zawierające docelowy adres jest zlokalizowane na stronie zerowej. Przykładem jest: „LDA (\$40,X)”.

O wiele częściej używany jest tryb *adresowania pośredniego indeksowego*, zwanego również *adresowaniem pośrednim postindeksowanym* (ang.: *indirect indexed*). Docelowy argument operacji pobierany jest ze zwiększonej o wartość rejestru Y lokalizacji wskazywanej przez dwa bajty adresu znajdujące się w podanym miejscu strony zerowej. Przykładem jest: „LDA (\$40),Y”.

Listę rozkazów procesora 6502 zawiera niniejsza tabela:

ADC	Dodaj do akumulatora (z przeniesieniem).
AND	Wykonaj logiczne AND z akumulatorem.
ASL	Wykonaj arytmetyczne przesunięcie w lewo. Bit0=0 C=Bit7.
BCC	Wykonaj skok kiedy flaga C skasowana.
BCS	Wykonaj skok kiedy flaga C ustawiona.
BEQ	Wykonaj skok kiedy porównane wartości jest równe (zero).
BIT	Testuj bity z bitami akumulatora.
BMI	Wykonaj skok jeżeli rezultat operacji jest ujemny.
BNE	Wykonaj skok kiedy porównanie nie jest równe (nie jest zerem).
BPL	Wykonaj skok jeżeli rezultat jest dodatni.
BRK	Wykonaj przerwanie programowe BREAK.
BVC	Wykonaj skok kiedy flaga V jest skasowana.
BVS	Wykonaj skok kiedy flaga V jest ustawiona.
CLC	Wyczyść flagę C.
CLD	Wyłącz tryb dziesiętny.
CLI	Skasuj bit wyłączenia przerwań.
CLV	Skasuj flagę V.
CMP	Porównaj z wartością akumulatora.
CPX	Porównaj z wartością rejestru X.
CPY	Porównaj z wartością rejestru Y.
DEC	Zmniejsz wartość o 1.
DEX	Zmniejsz wartość rejestru X o 1.
DEY	Zmniejsz wartość rejestru Y o 1.
EOR	Wykonaj logiczne XOR z akumulatorem.
INC	Zwiększ wartość o 1.
INX	Zwiększ wartość rejestru X o 1.
INY	Zwiększ wartość rejestru Y o 1.
JMP	Wykonaj skok pod podany adres.
JSR	Wykonaj skok do procedury, zachowując adres powrotu.
LDA	Ustaw wartość akumulatora.
LDX	Ustaw wartość rejestru X.
LDY	Ustaw wartość rejestru Y.
LSR	Wykonaj logiczne przesunięcie w prawo. Bit7=0 C=Bit0.
NOP	Nie rób nic.
ORA	Wykonaj logiczne OR z akumulatorem.
PHA	Umieść wartość akumulatora na stosie.
PHP	Umieść rejestr flag procesora na stosie.

PLA	Pobierz ze stosu wartość i umieść ją w akumulatorze.
PLP	Pobierz ze stosu rejestr flag procesora.
ROL	Wykonaj bitowe przesunięcie w lewo. C=Bit7 Bit0=C.
ROR	Wykonaj bitowe przesunięcie w prawo. C=Bit0 Bit7=C.
RTI	Powrót z przerwania.
RTS	Powrót z procedury.
SBC	Odejmij wartość od akumulatora (z pożyczką)
SEC	Ustaw flagę C.
SED	Ustaw tryb dziesiętny.
SEI	Ustaw bit wyłączenia przerw.
STA	Zapisz wartość akumulatora w pamięci.
STX	Zapisz wartość rejestru X w pamięci.
STY	Zapisz wartość rejestru Y w pamięci.
TAX	Skopiuj wartość akumulatora do rejestru X.
TAY	Skopiuj wartość akumulatora do rejestru Y.
TSX	Skopiuj wartość rejestru stosu do rejestru X.
TXA	Skopiuj wartość rejestru X do akumulatora.
TXS	Skopiuj wartość rejestru X do rejestru stosu.
TYA	Skopiuj wartość rejestru Y do akumulatora.

Tabela 8.2. Lista rozkazów procesora 6502

Źródło: Opracowane na podstawie: D. Eyes, R. Lichty, Programming the 65816 Including the 6502, 65C02 and 65802, New York 1986, Prentice Hall Press

#### 8.2.4. Tworzenie pliku binarnego w formacie relokalnym XEUX

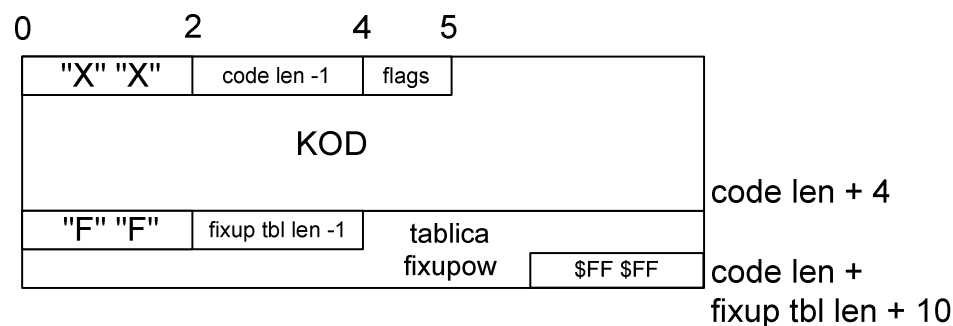
Ze względu na brak odpowiednich aplikacji napisanych dla systemu XEUX, do przygotowywania pliku wykonywalnego w relokalnym formacie akceptowanym przez system należy wykorzystać dowolny assembler generujący kod dla procesora 6502, oraz drobny program narzędziowy MAKRELA napisany w języku Turbo Basic XL.

Aby utworzyć plik w odpowiednim formacie należy zasemblować kod źródłowy procesu dwukrotnie, pierwszy raz pod adres docelowy 0x1000, drugi raz pod adres 0x2000. Segment kodu powinien być ciągły. Jeżeli wykorzystywany assembler generuje kod w postaci wielu segmentów, jak robi m.in. assembler z pakietu MAE, należy przed użyciem MAKRELI scalić segmenty np. z wykorzystaniem załączonego programu UNIFY.EXE.

Ścieżkę do binarnych plików zawierających efekt asemblacji należy następnie podać programowi MAKRELA jako odpowiednio SOURCE1\$ i SOURCE2\$, w zmiennej DEST\$

natomiast ścieżkę do pliku wynikowego, jaki ma zostać utworzony. Po uruchomieniu (RUN) program wypisze długość kodu, kolejne wartości odczytywane z obu plików źródłowych (oraz ich reprezentację w formie znaków ATASCII), a następnie tablicę fixupów – czyli miejsc w kodzie w postaci indeksu w których należy zmodyfikować adres o wartość strony pamięci przydzielonej dla początku procesu – wartość tą znajdziemy w tablicy procesu w zmiennej PR.PAGE\_. W celu umożliwienia weryfikacji poprawności, podczas zapisywania pliku wyświetlane są wartości odczytywane z SOURCE1\$ i zapisywane do DEST\$, a punkty podlegające modyfikacji podczas relokacji oznaczane dodatkowo gwiazdką.

Format pliku relokowalnego zilustrowany jest na poniższym rysunku:



Rys. 8.3. Format binarnego pliku relokowalnego XEUX

Źródło: opracowanie własne

Format XX zawiera obowiązkowe jednobajtowe pole flag. Umożliwia ono przekazywanie systemowi informacji o dodatkowych blokach, jak np. informacje o dodatkowej przestrzeni na dane, bloki zapewniające specjalne traktowanie, etc. Zapalony bit 7 pola flag wskazuje obecność następującego bezpośrednio po bloku kodu bloku fixupów. Blok fixupów zakończony jest dwoma bajtami \$FF.

MAKRELA.LST:

```

10 DIM FIKSAPY(1000)
15 DIM SOURCE1$(30), SOURCE2$(30), DEST$(30)
20 SOURCE1$="D:PROC3.10"
25 SOURCE2$="D:PROC3.20"
30 DEST$="D:PROC3.XX"
35 OPEN #1,4,0,SOURCE1$
40 OPEN #2,4,0,SOURCE2$
45 FOR I=1 TO 2
50   GET #1,A

```

```

55   IF A<>255 THEN GOTO 440
60  NEXT I
65  FOR I=3 TO 5
70    GET #1,A
75  NEXT I
80  B=A
85  GET #1,A
90  B=B+(A-16)*256
95  ? :? "Dlugosc kodu: ";B:?
100 CLOSE #1
105 OPEN #1,4,0,SOURCE1$
110 FOR I=0 TO 5
115   GET #1,RAZ
120   GET #2,DWA
125  NEXT I
130 FOR I=0 TO B
135   GET #1,RAZ
140   GET #2,DWA
145   ? I;" ";RAZ,DWA,"<";CHR$(RAZ),"<";CHR$(DWA)
150   IF RAZ<DWA
155     FIK=FIK+1
160     FIKSAPY(FIK)=I
165   ENDIF
170 NEXT I
175 ? :? "Fiksapy: ";
180 FOR I=1 TO FIK
185   ? FIKSAPY(I);" ";
190 NEXT I
195 CLOSE #1:CLOSE #2
200 OPEN #1,4,0,SOURCE1$
205 OPEN #2,8,0,DEST$
210 FOR I=0 TO 3
215   GET #1,RAZ
220 NEXT I
225 PUT #2,ASC("X")
230 PUT #2,ASC("X")
235 IF FIK>0
240   REM EXT bit 7 - fixups segment
245   PUT #2,128
250 ENDIF
255 GET #1,RAZ
260 PUT #2,RAZ
265 GET #1,RAZ
270 PUT #2,RAZ-16
275 PRINT
280 FOR I=2 TO B+2
285   GET #1,RAZ:? I,RAZ;" ";
290   TU=0
295   IF FIK>0:FOR J=1 TO FIK
300     IF FIKSAPY(J)=I-2 THEN TU=1
305     NEXT J
310   ENDIF
315   IF TU=1

```

```

320     PUT #2,RAZ-16:? RAZ-16,"*"
325     ELSE
330     PUT #2,RAZ:? RAZ;" "
335     ENDIF
340 NEXT I
345 IF FIK>0
350     REM naglowek: "FF"
355     REM + dlugosc tabeli fixupow -1
360     PUT #2,ASC("F")
365     PUT #2,ASC("F")
370     PUT #2,((FIK*2)-1) MOD 256
375     PUT #2,((FIK*2)-1) DIV 256
380     FOR I=1 TO FIK
385         PUT #2,FIKSAPY(I) MOD 256
390         PUT #2,FIKSAPY(I) DIV 256
395     NEXT I
400 ENDIF
405 CLOSE #1
410 PUT #2,255
415 PUT #2,255
420 CLOSE #2
425 ? :? :? "Zapisano: ";
430 ? DEST$:?
435 END
440 ? "Source files are not binary!"
445 CLOSE #1
450 CLOSE #2

```

### 8.2.5. Kody błędów

Procedury systemowe sygnalizują pojawienie się błędu w dwojaki sposób. Po pierwsze, przed powrotem do programu wywołującego ustawiają flagę Carry procesora. Umożliwia to łatwą detekcję błędu, przykład ilustruje poniższy kod:

```

...
JSR PROCEDURA_SYSTEMOWA_      ; wywołanie funkcji
BCC ?OK                         ; test flagi Carry
...kod obsługi błędu...
?OK ...ciąg dalszy programu...

```

W ten sposób za pomocą dwubajtowej instrukcji BCC (z ang. *Branch if Carry Clear*) ominąć można kod obsługi błędu jeżeli takowy nie wystąpił, bądź też zrealizować jego obsługę w przypadku przeciwnym. Drugi sposób sygnalizacji błędu, uzupełniający sposób pierwszy, to kod błędu zwracany w rejestrze indeksowym X. Jeżeli wartość rejestru po wyjściu z procedury systemowej wynosi zero, oznacza to, iż funkcja zakończyła swoje

działanie z sukcesem. Wartość różna od zera oznacza wystąpienie jednego z opisanych w tabeli błędów:

Kod błędu	Funkcja zwracająca	Opis
\$FF	MALLOC_	„not enough free pages” : funkcja nie była w stanie znaleźć ciągłej przestrzeni wolnej pamięci w podanej ilości stron.
\$FE	PRLOAD_	„maximum number of processes reached”: brak wolnego miejsca na stronę bazową procesu, osiągnięto maksymalną dla systemu liczbę procesów.
\$FD	PRLOAD_	„not a XEUX binary file”: wczytany plik nie jest w formacie binarnym XEUX
\$FC	PRLOAD_	„fixup table missing”: we wczytanym pliku nie znaleziono tablicy fixupów
\$FB	MALLOC_	„number of requested pages too big”: wielkość przestrzeni do zaalokowania w pamięci przekracza 63 strony (16128 bajtów).
\$FA	KILL_	„process number out of range”: podany numer procesu przekracza wartość dopuszczalnej ilości procesów w systemie.
\$F9	KILL_	„process does not exist”: proces o podanym numerze nie istnieje
\$F8	KILL_	„higher priority signal waiting”: w kolejce dla danego procesu oczekuje już sygnał o wyższym priorytecie.
\$F7	KILL_	„cannot send to idle process”: próba wysłania sygnału do procesu zerowego
\$F6	KILL_	„signal number out of range”: próba wysłania sygnału spoza dopuszczalnego zakresu.
\$F5	SIGIVEC_	„signal not supported”: próba rejestracji procedury obsługi sygnału o numerze nie wspieranym w aktualnej wersji systemu (dotyczy to wszystkich sygnałów z wyłączeniem 1, 15, 23, 30 i 31).
\$F4	PRREAD_	„binary too long”: długość całkowita wczytywanego pliku binarnego przekracza długość bufora.
\$F3	FORK_	„tty busy”: konsola wskazana dla nowego procesu jest już zajęta
\$F2	FORK_	„unable to open tty”: próba otwarcia konsoli o podanym numerze nieudana; prawdopodobnie został podany numer konsoli nieistniejącej w systemie.

Tabela 8.3. Wykaz kodów błędów

Źródło: opracowanie własne

## WYKAZ RYSUNKÓW

Rys. 3.1. Rozwój komputerów domowych Atari .....	12
Rys. 3.2. <i>Montezuma's Revenge</i> autorstwa R. Jaegera .....	13
Rys. 3.3. Rozklokowanie grup na mapie Polski w okresie rozkwitu sceny – rok 1996 .....	14
Rys. 3.4. Budowa mikroprocesora 6502 .....	17
Rys. 3.5. Znaczenie bitów rejestru PORTB w Atari 130XE .....	19
Rys. 3.6. <i>Numen</i> autorstwa Taquart – ekran tytułowy .....	29
Rys. 3.7. Znaczenie bitów rejestru PORTB w <i>IMB Pasiu SIMM Expansion</i> .....	30
Rys. 3.8. Demonstracja możliwości prototypu karty <i>Video Board XE</i> .....	33
Rys. 4.1. Organizacja procedur w ramach modułów .....	36
Rys. 4.2. Elementy składowe pliku wynikowego systemu .....	37
Rys. 4.3. Proces tworzenia plików wykonywalnych dla systemu XEUX .....	38
Rys. 5.1. Organizacja pamięci banku w trybie przydziału pamięci „bank per proces” .....	42
Rys. 5.2. Organizacja pamięci w trybie przydziału z wykorzystaniem podbanków .....	42
Rys. 5.3. Schemat blokowy procedury przydziału pamięci <i>MALLOC_</i> .....	44
Rys. 5.4. Organizacja pamięci banku XMS w trybie przydziału liniowego .....	45
Rys. 5.5. Procedura dostępu do danych wolnych .....	46
Rys. 5.6. Diagram stanów procesu .....	47
Rys. 5.7. Struktura pola <i>PR.STATE_</i> .....	48
Rys. 5.8. Znaczenie bitów pola <i>RUNLEVEL_</i> .....	50
Rys. 5.9. Schemat blokowy planisty krótkoterminowego – procedura główna .....	51
Rys. 5.10. Schemat blokowy planisty krótkoterminowego – procedury powrotu .....	52
Rys. 5.11. Schemat blokowy planisty krótkoterminowego – <i>SKE_MLTI_</i> część 1 .....	53
Rys. 5.12. Schemat blokowy planisty krótkoterminowego – <i>SKE_MLTI_</i> część 2 .....	54
Rys. 5.13. Schemat blokowy planisty krótkoterminowego – <i>SKE_IOC_</i> .....	55
Rys. 5.14. Schemat blokowy planisty krótkoterminowego – <i>SKE_SIG_</i> .....	56
Rys. 5.15. Procesy wykorzystujące współdzielone dane zabezpieczane semaforem .....	58
Rys. 5.16. Schemat blokowy planisty krótkoterminowego – <i>SKE_TRM_</i> i <i>SKE_TER_</i> .....	59
Rys. 5.17. Organizacja pamięci dla konsol wirtualnych w XEUX .....	61
Rys. 5.18. Tablica kodów ATASCII podstawowego zestawu znaków .....	63
Rys. 5.19. Model uproszczony obsługi procesów przez planistę krótkoterminowego .....	64
Rys. 5.20. Procedura korzystania z CIO z wykorzystaniem semaforów .....	65
Rys. 5.21. Organizacja przekazywanych parametrów na stosie procesu potomnego .....	67
Rys. 5.22. Struktura stosu w momencie przekazania kontroli procedurze obsługi sygnału ....	68
Rys. 5.23. Wykorzystanie mechanizmu sygnałów do realizacji skoków warunkowych .....	69
Rys. 6.1. Powłoka systemowa .....	72
Rys. 6.2. Program PS w działaniu .....	73



Rys. 6.3. Program KILL w działaniu.....	74
Rys. 6.4. Monitor systemowy.....	75
Rys. 6.5. Program WRITE – wprowadzanie wiadomości .....	76
Rys. 6.6. Program WRITE – wiadomość odebrana.....	77
Rys. 8.1. Organizacja przestrzeni adresowej Atari XE pod kontrolą XEUX.....	84
Rys. 8.2. Organizacja strony zerowej .....	85
Rys. 8.3. Format binarnego pliku relokalnego XEUX.....	92

## WYKAZ TABEL

Tabela 3.1. Znaczenie bitów rejestru flag procesora .....	18
Tabela 3.2. Przerwania IRQ w kolejności priorytetów .....	21
Tabela 3.3. Struktura IOCB.....	26
Tabela 3.4. Struktura tablicy adresowej sterownika.....	26
Tabela 3.5. Struktura nagłówka rozruchu ( <i>boot</i> ).....	28
Tabela 5.1. Struktura strony bazowej procesu .....	40
Tabela 5.2. Znaczenie bitów 0-3 pola PR.STATE_ .....	48
Tabela 5.3. Zawartość tablicy TSTTBL_.....	62
Tabela 5.4. Podzbiór standardowych sygnałów wybrany do implementacji w XEUX.....	67
Tabela 6.1. Lista komend dla <i>barszcz 0.15</i> .....	71
Tabela 8.1. Zmienne systemowe z zakresu 0x80-0xBF przydatne dla programisty .....	86
Tabela 8.2. Lista rozkazów procesora 6502.....	91
Tabela 8.3. Wykaz kodów błędów .....	95

## BIBLIOGRAFIA

Wydawnictwa zwarte:

1. W. Zientara, *Mapa pamięci Atari XL/XE – Dyskowe systemy operacyjne*, Warszawa 1988, Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej
2. W. Zientara, *Mapa pamięci Atari XL/XE – Podstawowe procedury systemu operacyjnego*, Warszawa 1988, Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej
3. W. Zientara, *Mapa pamięci Atari XL/XE – Procedury wejścia/wyjścia*, Warszawa 1988, Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej
4. W. Stallings, *Systemy operacyjne*, Wrocław 2004, Robomatic
5. J. Ruszczyk, *Asembler 6502*, Warszawa 1987, Stołeczny Ośrodek Elektronicznej Techniki Obliczeniowej
6. M. J. Bach, *Budowa systemu operacyjnego UNIX®*, Warszawa 1995, Wydawnictwa Naukowo-Techniczne
7. H. Kruszyński, K. Kulpa, *Mikroprocesor 6502 i jego rodzina*, Warszawa 1989, Wydawnictwo Czasopism i Książek Technicznych NOT-SIGMA
8. A. Silberschatz, P. B. Galvin, *Podstawy systemów operacyjnych*, Warszawa 2002, Wydawnictwa Naukowo-Techniczne
9. D. Eyes, R. Lichty, *Programming the 65816 Including the 6502, 65C02 and 65802*, New York 1986, Prentice Hall Press
10. I. Chadwick, *Mapping The Atari – Revised Edition*, Greensboro 1985, COMPUTE! Publications, Inc.
11. J. Stanton, D. Pinal, *Atari Graphics & Arcade Game Design*, Los Angeles 1984, Arrays, Inc.
12. Ch. Crawford, L. Winner, J. Cox, A. Chen, J. Dunion, K. Pitta, B. Fraser, G. Makrea, *De Re Atari – A Guide to Effective Programming*, 1982, Atari, Inc.
13. M. Andrews, *Atari Roots*, Chatsworth 1984, Datamost, Inc.

Czasopisma i magazyny:

1. *Tajemnice Atari*, 1991-1993
2. „Syzygy” – magazyn użytkowników i sympatyków ośmiobitowego Atari, numery 3-8
3. *Serious Magazine*, numery 1-16
4. *Antic*, numery

Artykuły, strony internetowe oraz inne zasoby:

1. J. Bernašek, *The Inside of Network Games*, BEWESOFT
2. J. Potter, *Game Link II sample code*, 1997
3. The AGDA Group, *GameLink-II Specification*, DataQue Software, 1993
4. *Atariki* – *atari.area* Wikipedia, <http://atariki.krap.pl/>
5. *atari.area*, <http://atariarea.krap.pl/>
6. *Atari History Museum*, <http://www.atarimuseum.com/>
7. *Jindroush.ATARI.org*, <http://krap.pl/mirrorz/atari/www2.asw.cz/kubecj/>
8. *Mathy's new and improved homepage*, <http://www.mathyvannisselroy.nl/>
9. *Wikipedia, the free encyclopedia*, <http://en.wikipedia.org/>
10. *Das Atari Museum*, <http://www.atari-museum.de/>
11. *Pasia Stołówka*, <http://hardware.atari8.info/>
12. S.T.A.R., *Atari Cuida Tu Salud*, Matranet, Issue #03 – Febrero 2002
13. electron/taquart, *Video Board XE specification*,  
<http://atariarea.krap.pl/pliki/rozne/vbxe.pdf>
14. *FreeBSD Library Functions Manual*
15. *The UNIX System*, <http://www.unix.org/>
16. *Numen: The popular 8bit demo by TAQUART*, <http://numen.scene.pl/>
17. *Project F7*, <http://pasiu.krap.pl/>