rozdział drugi

KOPROCESOR

W tym rozdziale nauczysz się jak używać koprocesor graficzny Amigi i jego uproszczony zestaw instrukcji do zarządzania rejestrami kontrolującymi obraz w czasie pionowego wygaszenia. Pokażemy jak układać instrukcje Coppera w copperlisty (programy wykonywane przez ten koprocesor), jak używać copperlist w trybie z przeplotem oraz jak używać Coppera razem z blitterem. Rozdział ten zajmie się podstawowymi informacjami o Copperze. Kolejne rozdziały, opisujące playfieldy, sprite'y, dźwięk i blitter, będą zawierać więcej szczególnych przypadków użycia Coppera.

O układzie Copper

Copper to koprocesor ogólnego przeznaczenia, będączy częścią jednego z układów specjalizowanych Amigi (*Agnus* w OCS, *Super Agnus* w ECS i *Alice* w ukladach AGA). Otrzymuje instrukcje przy użyciu jednego z kanałów bezpośredniego dostępu do pamięci (DMA – ang. *Direct Memory Access*). Copper może kontrolować niemalże cały system graficzny Amigi, pozwalając procesorowi 680x0 na wykonywaniu logiki programu. Copper ma także bezpośredni dostęp do większości rejestrów sprzętowych. To bardzo potężne narzędzie umożliwiające sprawne zarządzanie zmianami rejestrów odpowiedzialnych za wyświetlany obraz, które to zmiany muszą być wykonywane w czasie pionowych wygaszeń. Do jego zadań należy kontrola nad aktualizacją zawartości rejestrów, zmiana pozycji sprite'ów, aktualizowanie kanałów dźwiękowych i kontrola nad blitterem.

Jedną z podstawowych funkcji Coppera jest możliwość odczekania (instrukcja WAIT) na konkretną pozycję wiązki wideo na ekranie, które prowadzi do odpowiedniego ustawienia (instrukcja MOVE) rejestru systemowego. W czasie wykonywania instrukcji WAIT Copper bezpośrednio sprawdza położenie wiązki wideo. Oznacza to, że Copper, czekając aż wiązka osiągnie zadaną pozycję na ekranie, nie używa w tym czasie szyny komunikacyjnej. Komunikacja jest więc dostępna dla każdego innedo kanału DMA lub dla procesora 680x0.

Kiedy warunek instrukcji WAIT zostanie spełniony, Copper przejmuje cykle zegarowe albo od blittera albo od procesora i przenosi zadane dane do wybranego rejestru sprzętowego.

Copper wysyła rządnia do szyny tylko w czasie nieparzystych cykli. Zapobiega to konfliktom z dźwiękiem, sprite'ami, dyskiem i większością innych kanałów DMA, które korzystają z parzystych cykli pamięci. Z tego powodu Copper potrzebuje priorytetu nad procesorem i blitterem (kanałem DMA który obsługuje animacje, rysowanie linii i wypełnianie wielokątów).

Tak jak w przypadku pozostałych kanałów DMA, Copper może wykonywać swój program i pobierać dane umieszczony wyłącznie w pamięci Chip.

Czym jest instrukcja Coppera?

Jako koprocesor, Copper dodaje swoje instrukcje do zestawu instrukcji procesora 680x0. Copper ma tylko trzy instrukcje ale dzięki nim można zrobić na prawdę wiele:

- WAIT – zaczekaj na pozycję wiązki na ekranie zadaną jako współrzędne X i Y.
- MOVE – umieść zadaną wartość w jednym z rejestrów sprzętowych.
- SKIP – pomiń następną instrukcję jeśli wiązka przekroczyła zadaną pozycję na ekranie.

Wszystkie instrukcje Coppera zawierają dwa 16-bitowe słowa umieszczone w pamięci jedno po drugim. Za każdym razem kiedy Copper pobiera instrukcję – pobiera oba słowa.

Instrukcje MOVE i SKIP wymagają dwóch cykli dostępu do pamięci, każda składa się z dwóch słów. Ponieważ Copper pobiera dane w cyklach nieparzystych, każda instruckja wymaga czasu równego czterem cyklom. Instrukcja MOVE wymaga trzech cykli co w rezultacie przekłada się na czas sześciu cykli.

Pomimo iż Copper ma bezpośredni dostęp tylko do rejestrów sprzętowych, może mieć pośrednio wpływ również na pamięć dzięki blitterowi. Więcej informacji o tym jak używać Coppera do kontrolowania blittera znajduje się w sekcji „Rejestr kontrolny” oraz „Użycie Coppera z blitterem”.

Poniżej opisano instrukcje WAIT i MOVE. Instrukcja SKIP została opisana w sekcji „Techniki zaawansowane”.

Instrukcja MOVE

Instrukcja MOVE przepisuje dane z pamięci RAM do rejestru przeznaczenia. Dana zawarta jest w drugim słowie instrukcji. Pierwsze słowo zawiera adres rejestru docelowego. Procedura przenoszenia danych jest opisana szczegółowo w sekcji „Podsumowanie informacji o instrukcjach Coppera”.

PIERWSZE SŁOWO INSTRUKCJI MOVE (IR1)

Bit 0	Zawsze wygaszony (0)
Bity 8 – 1	Adres rejestru docelowego (DA8 – 1)
Bity 15 – 9	Nie używane, powinny być wyczyszczone (0)

DRUGIE SŁOWO INSTRUKCJI MOVE (IR2)

Bity 15 – 0	16 bitowa dana, która zostanie zapisana w rejestrze docelowym

Copper może zapisywać dane w następujących rejestrach:

- Każdym rejestrze, którego adres jest większy niż $20.[1]
- Każdym rejestrze z zakresu $10 - $20 jeśli bit niebezpieczeństwa jest ustawiony (ma wartość 1). Bit niebezpieczeństwa Coppera znajduje się w rejestrze kontroli Coppera COPCON, opisanym w sekcji „Rejestr kontrolny”.
- Copper nie może zapisywać danych do rejestrów z adresów poniżej $10.

W dodatku B znajduje się lista wszystkich rejestrów sprzętowych.

Poniższy przykład instrukcji MOVE ustawia wskaźnik bitplanu 1 na adres $21000 a wskaźnik bitplanu 2 na adres $25000.[2]

```
DC.W  $00E0,$0002       ; umieść $0002 w rejestrze $0E0 (BPL1PTH)
DC.W  $00e2,$1000       ; umieść $1000 w rejestrze $0E0 (BPL1PTL)
DC.W  $00E4,$0002       ; umieść $0002 w rejestrze $0E0 (BPL2PTH)
DC.W  $00E6,$5000       ; umieść $5000 w rejestrze $0E0 (BPL2PTL)
```

1 Liczby szesnastkowe w odróżnieniu od dziesiętnych poprzedza znak „$”.
2 Wszystkie przykłady kodu źródłowego są w języku assemblera.

Odpowiednie pliki „.i” są dołączane (ang. *include*), dzięki czemu w kodzie można używać nazw rejestrów sprzętowych zamiast ich adresów. Uzywanie nazw rejestrów dzięki systemowym plikom wsadowym jest szczególnie zalecane. Pozwoli to na łatwą adaptację kodu źródłowego do przyszłych usprawnień sprzętowych. Dla przykładu:

```
INCLUDE „hardware/custom.i”
DC.W  bplpt+$00,$0002   ; umieść $0002 w rejestrze $0E0 (BPL1PTH)
DC.W  bplpt+$e2,$1000   ; umieść $1000 w rejestrze $0E0 (BPL1PTL)
DC.W  bplpt+$E4,$0002   ; umieść $0002 w rejestrze $0E0 (BPL2PTH)
DC.W  bplpt+$E6,$5000   ; umieść $5000 w rejestrze $0E0 (BPL2PTL)
```

Dla czelów tego podręcznika przygotowaliśmy specjalny plik wsadowy (zobacz dodatek I), który definiuje wszystkie nazwy rejestrów sprzętowych bazując na pliku „hardware/custom.i”. Dzięki niemu wszystkie przykłady będą łątwiejsze do odczytania i zrozumienia. Większosć zawartych w tej książce przykładów znalazło się tu po to, aby wyjaśnić w jaki sposób wszystko działa na poziomie sprzętowym i nie nadaje się do bezpośredniego użycia bez dodatkowych modyfikacji i dodatkowego kodu.

Instrukcja WAIT

Instrukcja WAIT zmusza Copper do odczekania, aż wiązka wideo osiągnie lub przekroczy zadaną w instrukcji pozycję na ekranie. W chwili kiedy Copper czeka, nie wykonuje żadnych innych operacji.

Pierwsze słowo instrukcji zawiera pionową i poziomą współrzędną pozycji wiązki. Drugie słowo zawiera maskę, określającą które bity pozycji wiązki braż pod uwagę podczas porównania.

PIERWSZE SŁOWO INSTRUKCJI WAIT (IR1)

Bit 0	Zawsze włączony (1)
Bity 15 – 8	Pozycja wiązki w pionie (VP)
Bity 7 – 1	Pozycja wiązki w poziomie (HP)

DRUGIE SŁOWO INSTRUKCJI WAIT (IR2)

Bity 0	Zawsze skasowany (0)
Bit 15	Bit blittera. Zazwyczaj włączony (1) (Zobacz „Techniki zaawansowane”)
Bity 14 – 8	Bity maski pozycji pionowej (VE)
Bity 7 – 1	Bity maski pozycji poziomej (HE)

Poniższy przykład instrukcji WAIT czeka na linię 150 ($96) z poziomą pozycją „wymaskowaną”.

```
DC.W  $9601,$FF00       ; zaczekaj na linię 150
                        ; zignoruj pozycję poziomą
```

Następny przykład instrukcji WAIT czeka na linię 255 i pozycję poziomą 254. Takie zdarzenie nigdy nie zajdzie więc Copper w tym momencie kończy pracę aż do czasu następnego wygaszenia pionowego.

```
DC.W  $FFFF,$FFFE       ; zaczekaj na linię 255
                        ; H = 254 (kończy copperlistę)
```

Aby zrozumieć dlaczego pozycja VP=$FF HP=$FE nigdy nie może zostać osiągnięta musimy popatrzeć na operację porównaina i ograniczenia rozmiaru informacji o pozycji. Linia 255 jest osiągalna. Prawdę mówiąc to linia 255 jest jednocześnie maksymalną wartością jaką można zapisać na bitach pozycji pionowej. Następna linia na ekranie spowoduje, że licznik „przekręci” się i osiągnie wartość 0 (linia 256 dla operacji porównania będzie miała wartość 0). Numer linii nigdy

nie będzie większy niż $FF. Pozycja pozioma osiąga maksymalną wartość $E2. Oznacza to, że największą wartością jaka może pojawić się w operacji porównania będzie $FFE2. Oczekując więc na pozycję $FFFE linia $FF zostnie osiągnięta ale pozycja pozioma $FE – nie. Tak więc pozycja $FFFE nigdy nie zostanie osiągnięta.

Mogłoby się wydawać, że wystarczy więc poczekać tylko na pozycję poziomą równą $FE, podając numer linii mniejszy niż $FF lub wręcz 0. Nic bardziej mylnego. Operacja porównania zwraca rezultat po osiągnięciu lub przekroczeniu danej pozycji. Jeśli pionowa pozycja jest mniejsza niż $FF to w momencie kiedy pozycja wiązki osiągnie linię większą niż porównywana, operacja porównania zwróci wartość „true” i zakończy się.

Poniższe informacje o poziomej i pionowej pozycji wiązki wideo dotyczą zarówno instrukcji WAIT jak i SKIP. Instrukcja SKIP opisana jest w sekcji „Techniki zaawansowane”.

POZIOMA POZYCJA WIĄZKI

Pozioma pozycja promienia wizji może przyjmować wartość od 0 do $E2. W czasie operacji porównania najmniej znaczący bit nie jest brany pod uwagę. Mamy więc 113 pozycji, które mają znaczenie dla Coppera. Odpowiada to czterem pikselom w niskiej albo ośmiu w wysokiej rozdzielczości. Wygaszanie pionowe obejmuje zakres od $0F do $35. Standardowy ekran o szerokości 320 pikseli ma nieużywaną część z zakresu od $04 do $47, która jest zapełniona kolorem tła.

W systemie NTSC linie nie są jednakowej długości. Ekran jest zapełniany naprzemiennie przez tak zwane długie i krótkie linie (długie to 228 taktów zegara zmiany koloru, 0 - $E3 a krótkie – 227 taktów). W systemie PAL wszystkie linie mają długość 227 taktów. Dla systemu wyświetlania wszystkie linie mają długość 227 1/2. Natomiast Copper potrafi odróżnić długie i krótkie linie.\

PIONOWA POZYCKA WIĄZKI

Pionowa pozycja wiązki odpowaida poszczególnym liniom. Może przyjąć maksymalną wartość 255. W rzeczywistości NTSC wyświetla 262 linie (312 sla PAL). Wydawać by się mogło nieosiŋalnym wykorzystanie tych ostatnich szesciu lub siedmiu linii. Mamy przecież tylko 8 bitów, czyli maksymalnie 256 wartosci opisujących pionową pozycję. Poniżej przedstawiono jednen z najprostrzych sposobów na obejście tego ograniczenia.

Instrukcja Coppera	**Wyjaśnienie**
WAIT – zaczekaj na pozycję (0,255)	*W tym momencie licznik pozycji pionowej „przekręca” się osiągając ponownie wartość 0.*
WAIT – zaczekaj na pozycję z zakresu 0 – 5, co pokrywa ostatnie 6 linii zanim rozpocznie się pionowe wygaszenie.	*W rezultacie mamy 256 + 6 = 262 linie jakie pokonuje wiązka w trakcie których Copper może wykonywać swoje instrukcje.*

Zauważmy, że pion jest podobny do poziomu. Mamy krótkie i długie linie ale w trybie z przeplotem mamy też długie i krótkie klatki (ramki). Dla systemu NTSC mamy odpowiednio 262 i 263 linie, w systemie PAL – 312 i 313 linii. Ta naprzemienność linii i ramek daje nam standardowy powtarzający się wzorzec 4 klatek w NTSC:

krótka ramka kończy się krótką linią
długa ramka kończy się długą linią
krótka ramka kończy się długą linią
długa ramka kończy się krótką linią

Zwiększenie poziomej pozycji o jeden zajmuje jeden taks zegara systemowego (procesor prasuje w rytmie dwutaktowym).

NTSC – 3 579 545 Hz
PAL – 3 546 895 Hz
tryb genlocka – standardowa wartość zegara plus / minus 2%

BITY PORÓWNANIA

Bity 14 – 1 najczęściej są ustawiane (wartość 1). Użycie tych bitów opisane jest w sekcji „Techniki zaawansowane”.

Używanie rejestrów Coppera

Copper posiada własny zestaw rejestrów odpowiedzialnych za jego pracę:

- rejestry lokacji
- rejstry adresów skoków
- rejestr kontroli

REJESTRY LOKACJI

Copper posiada dwa zestawy rejestrów lokacji:

COP1LCH	Najwyższe trzy bity adresu pierwszej copperlisty
COP1LCL	Niskie 16 bitów adresu pierwszej copperlisty
COP2LCH	Najwyższe trzy bity adresu drugiej copperlisty
COP2LCL	Niskie 16 bitów adresu drugiej copperlisty

W przypadku bezpośredniego dostępu do rejestrów sprzętowych bardzo często spotkasz się z koniecznością zapisu adresu do prady rejestrów. Rejestr o niższym adresie, którego nazwa często kończy się literą „H” (ang. *high*) oczekuje bardziej zanczącego słowa adresu. W przypadku wszystkich rejestrów sprzętowych słowo to będzie zawierać tylko trzy bity. Wynika to z faktu, że kanały DMA mogą operować tylko na pamięci Chip, której najwyższym adresem jest $7FFFF (512 kB, nowsze chipsety podnoszą to ograniczenie). Rejestr o adresie wyższym, z literą „L” kończącą jego nazwę, oczekuje mniej znaczące słowo adresu (15 bitów – tylko parzyste adresy).

Pisząc program wystarczy jednak zapisać cały adres jednorazowo posługując się adresem rejestru z literą „H”. Instrukcja zapisująca 18-bitowy adres umieści oba słowa poprawnie we właściwych lokacjach pamięci. W związku z tym, dla uproszczenia, rejestry lokacji Coppera będą nazywane COP1LC i COP2LC.

Rejestry lokacji Coppera przechowują dwa pośrednie adresy skoków używane przez Coppera. Układ ten pobiera kolejne instrukcje swojego programu posługując się własnym licznikiem programu, zwiększając go każdorazowo po pobraniu każdej instrukcji. W chwili zapisania odpowiedniego rejestru skoku (zwanego również adresem strobującym) adres z odpowiadającego mu rejestru lokacji jest ładowany do licznika programu Coppera. W wyniku tego Copper „przeskakuje” do nowej lokacji skąd pobierze swoją następną instrukcję. Pobieranie instrukcji odbywa się dalej sekwencyjnie aż do momentu, kiedy praca Coppera zostanie przerwana przez kolejne zapisanie adresu strobującego.

> *Restartowanie Coppera*. Na początku każdego interwału wygaszania pionowego, rejestr COP1LC jest automatycznie użyty do uruchomienia licznika programu. Oznacza to, że niezależnie od tego, co Copper robi w danej chwili, gdy nastąpi koniec wygaszania pionowego, Copper ponownie rozpoczyna swoją pracę od adresu z rejestru COP1LC.

ADRES REJESTRU SKOKU

W chwili zapisania rejestru skoku dowolną wartością, Copper przeładowuje swój licznik programu zawartością odpowiedniego rejestru lokacji. Copper sam może zapsywać dane do rejestrów lokacji i rejestrów skoków aby uzyskać w ten sposób coś na wzór instrukcji skoku. Przykładowo, może użyć

MOVE do zmiany adresu w rejestrze COP2LC. Następnie używając dowolnej danej w instrukcji MOVE zapisać adres strobujący COPJMP2 aby Copper przełączył się na nową copperlistę, ładując do licznika programu adres z rejestru COP2LC.

Są dwa rejestry skoków Coppera:

COPJMP1	Copper rozpoczyna pracę od adresu z rejestru COP1LC
COPJMP2	Copper rozpoczyna pracę od adresu z rejestru COP2LC

REJESTR KONTROLI

Copper ma stały dostęp tylko do niektórych rejestrów. Do innych może zapisywać dane tylko, kiedy specjalny bit kontroli jest ustawiony. Są też rejestry do których Copper nie ma dostępu. Rejestry do których Copper ma zawsze dostęp mają adresy od $80 do $FF włącznie (W dodatku B znajduje się lista wszystkich rejestrów uporządkowanych względem ich adresów). Te, do których nie ma dostępu leżą pod adresami od $00 do $3E. W tym zakresie leży też rejestr kontroli Coppera. Rejestry z zakresu od $40 do $7E są chronione wspomnianym już bitem znajdującym się właśnie w rejestrze kontroli.

W rejestrze kontroli Coppera, o nazwie COPCON, używany jest tylko bit 1. Bit ten, nazywany CDANG (ang. *Copper Danger Bit*), chroni wszystkie rejestry z zakresu od $40 do $7E. W tej grupie znajdują się m. in. rejestry kontrolujące blitter. Copper może do nich zapisywać dane jeśli bit CDANG jest ustawiony (ma wartość 1). Ochrona tych rejestrów przez ingerencją Coppera ma na celu ograniczenie możliwości nadpisania pamięci przez błędnie napisaną copperlistę.

Uwaga: Warto mieć na uwadze, że bit CDANG jest czyszczony po resecie.

Tworzenie listy instrukcji Coppera (copperlisty)

Lista instrukcji Coppera zawiera operacje ponownego ustawiania różnych rejestrów wykonywane w czasie wygaszania pionowego oraz instrukcje wpływające na zmiany widoczne na ekranie. Planując co się stanie w czasie wyświetlania każdej klatki, być może łatwiej będzie myśleć o każdej części składowej obrazu jako osobnym podsystemie: playfieldach, sprite'ach, dźwięku[3], przerwaniach, itp. Wtedy można zbudować osobną listę rzeczy, które muszą być zrobione dla każdego z tych podsystemów w chwili osiągnięcia przez wiązkę wideo założonych z góry pozycji.

Jeśli już stworzyłeś wszystkie niezbędne listy instrukcji dla każdego z podsystemów, łączysz je w jedną listę instrukcji, które zostaną wykonane przez Copper dla każdej klatki obrazu. Alternatywą jest ułożenie tej złączonej listy od razu za jednym podejściem, bez podziału na podsystemy.

Przykładowo, wkaźniki bitplanów używanych przez playfield wyświetlany na ekranie a także wskaźniki sprite'ów muszą być ustawione ponownie w czasie każdego wygaszenia pionowego, aby dane mogły zostać poprawnie pobrane za każdym razem, kiedy kolejna klatka obrazu zaczyna pojawiać się na ekranie. Do tego celu można użyć listy instrukcji Coppera, która robi poniższe:

WAIT – czekaj na pierwszą linię obrazu
MOVE – ustaw wskaźnik bitplanu 1
MOVE – ustaw wskaźnik bitplanu 2

3 Być może zastanawiasz się dlaczego dźwięk jest wymieniony jako część składowa obrazu (ang. each aspect of the display)? Ja też. Podejrzewam, że chodzi o „multimedialność" Amigi i fakt, że jej możliwości graficzne, dźwiękowe, itp. są traktowane jako nieodłączne składowe tego, co odbiorca (użytkownik) odczuwa różnymi zmysłami. Oczywiście wpływ na to ma fakt, że rejestry sprzętowe odpowiadają nie tylko za sam obraz, ale też dźwięk i inne rzeczy.

MOVE – ustaw wskaźnik sprite’a 1
... itd. ...

W innym przykładzie, kanały DMA dla sprite’ów, które tworzą ruchome obiekty, mogą być użyte kilkukrotnie w czasie wyświetlania tej samej klatki obrazu. Można zmienić rozmiar i kształt ponownie użytego sprite’a, aczkolwiek ponownie użyty sprite zazwyczaj ma ten sam zestaw kolorów. Zmiana kolorów przy ponownym użyciu sprite’a jest możliwa, w chwili kiedy Copper poczeka na wyświetlenie ostatniej linii sprite’a i zmieni kolory w rejestrzach przed wyświetleniem pierwszej linii zmienionego sprite’a o tym samym numerze.

WAIT – czekaj na pierwszą linię obrazu
MOVE – ustaw COLOR17
MOVE – ustaw COLOR18
MOVE – ustaw COLOR19
WAIT – czekaj na ostatnią linię + 1 pierwszego sprite’a w użyciu
MOVE – ustaw COLOR17
MOVE – ustaw COLOR18
MOVE – ustaw COLOR19, itd.

Tworząc copperlistę musisz pamiętać, żeby ostateczna jej postać była ułożona w kolejności, w jakiej wiązka wideo przemieszcza się po ekranie tworząc obraz. Wiązka „wędruje” po całym ekranie od punktu (0, 0) w górlny lewym rogu ekranu kończąc w dolnym prawym rogu ekranu w punkcie o współrzędnych (226, 262) dla NTSC lub (226, 312) dla PAL. Pierwsza liczba we współrzędnych oznacza pozycję X. Druga odpowiada za pozycję Y. Musimy więc pamiętać, że instrukcja dotycząca pozycji (0, 100) musi nastąpić po instrukcji dla pozycji (0, 60).

W związku ze sposobem działania instrukcji WAIT nie zawsze jest tak istotnym zachowanie kolejności instrukcji w ścisłym powiązaniu do pozycji promienia wizyjnego. Instrukja WAIT każe Copperowi oczekiwać aż wiązka osiągnie zadaną w instrukcji pozycję lub ją przekroczy.

Oznacza to, przykładowo, mając instrukcje jak poniżej:

WAIT – czekaj na pozycję (64, 64)
MOVE – zapisz dane

WAIT – czekaj na pozycję (60, 60)
MOVE – zapisz dane

że Copper wykona obie instrukcje MOVE mimo iż nie są one we właściwej kolejności. Warunek „większy niż” zapobiega „zawieszeniu się” Coppera jeśli wiązka przekroczyła zadaną pozycję. Wykonanie drugiej instrukcji MOVE będzie efektem ubocznym w poniższym przykładzie:

WAIT – czekaj na pozycję (60, 60)
MOVE – zapisz dane

WAIT – czekaj na pozycję (60, 60)
MOVE – zapisz dane

W czasie wykonywania drugiej instrukcji MOVE pozycja wiązki wideo wciąż będzie większa niż wartość zadana w parametrze, więc druga instrukcja też zostanie wykonana.

Zauważmy, że powyższa lista instrukcji powinna więc wyglądać tak

WAIT – czekaj na pozycję (60, 60)
MOVE – zapisz dane
MOVE – zapisz dane

ponieważ nic nie stoi na przeszkodzie, żeby po jednej instrukcji WAIT nastąpiło wiele instrukcji MOVE.

PRZYKŁADOWA KOMPLETNA COPPERLISTA

Na poniższym przykładzie pokazano kompletną listę instrukcji Coppera. Listę stworzono dla dwóch bitplanów, jeden pod adresem $21000 a drugi pod $25000. Na poczatku ekranu ładowane są rejestry kolorów poniższymi wartościami:

Rejestr	**Kolor**
COLOR00	biały
COLOR01	czerwony
COLOR02	zielony
COLOR03	niebieski

W linii 150, rejestry kolorów są przeładowywane innymi wartościami:

Rejestr	**Kolor**
COLOR00	czarny
COLOR01	zółty
COLOR02	szarobłękitny
COLOR03	purpurowy

Poniżej kompletna copperlista.

```
; Uwagi: 1. Copperlist musi znajdować się w pamięci Chip.
;        2. Uzyte adresy bitplanów są tylko przykładowe, narzucone z góry.
;        3. Adresy rejestrów w instrukcjach MOVE są offsetami bazowego
;           adresu układów specjalizowanych.
;        4. Jak zawsze, przykłady w tej książce zakładają, że aplikacja przejęła
;           pełną kontrolę nad sprzętem i nie jest w konflikcie z systemem
;           operacyjnym, który może użyc tego sprzętu w tym samym czasie.
;        5. Wiele przykładów z góry zakłada jakieś adresy. W rzeczywistości
;           trzeba samodzielnie zarezerwować pamięć właściwego typu używając
;           np. funkcji systemowej AllocMem().
;        6. Jak już wspomniano wcześniej, przykładowe kody źródłowe mają pomóc
;           zrozumieć jak Amiga działa na poziomie sprzętowym.
;        7. Poniższe pliki wsadowe są używane przez wszystkie przykłady
;           w tym rozdziale.
;
      INCLUDE "exec/types.i"
      INCLUDE "hardware/custom.i"
      INCLUDE "hardware/dmabits.i"
      INCLUDE "hardware/hw_examples.i"

COPPERLIST:
;
; ustaw wskaźniki dla dwóch bitplanów
;
      DC.W BPL1PTH,$0002  ; Przenieś $0002 do rejestru $0E0 (BPL1PTH)
      DC.W BPL1PTL,$1000  ; Przenieś $1000 do rejestru $0E2 (BPL1PTL)
      DC.W BPL2PTH,$0002  ; Przenieś $0002 do rejestru $0E4 (BPL2PTH)
      DC.W BPL2PTL,$5000  ; Przenieś $5000 do rejestru $0E6 (BPL2PTL)
```

```
;
; Załaduj rejestry kolów
;
     DC.W COLOROO,$0FFF  ; biały do rejestru $180 (COLOROO)
     DC.W COLOROl,$0F00  ; czerwony do rejestru $182 (COLOROl)
     DC.W COLOR02,$00F0  ; zielony do rejestru $184 (COLOR02)
     DC.W COLOR03,$000F  ; niebieski do rejestru $186 (COLOR03)
;
; 2 bitplany nieskiej rozdzielczości
;
     DC.W BPLCONO,$2200  ; 2 bitplany, color na wyjściu dla A1000
;
; Czekaj na linię 150
;
     DC.W $9601,$FF00  ; czekaj na linię 150, ignoruj pozycję poziomą
;
; zmień kolory „od połowy” ekranu
;
     DC.W COLOROO,$0000  ; czarny do rejestru $0180 (COLOROO)
     DC.W COLOROl,$0FF0  ; żółtydo rejestru $0182 (COLOROl)
     DC.W COLOR02,$00FF  ; szaroniebieski do rejestru $0184 (COLOR02)
     DC.W COLOR03,$0F0F  ; purpurowy do rejestru $0186 (COLOR03)
;
; Zakończ copperlistę oczekując na niemożliwą pozycję na ekranie
;
     DC.W $FFFF,$FFFE  ; czekaj na linię 255, punkt = 254 (nieosiągalne)
```

Więcej o rejestrach kolorów dowiesz się z rozdziału trzeciego „Obsługa playfieldów”.

Uruchamianie i zatrzymywanie Coppera

URUCHAMIANIE COPPERA PO RESECIE

W czasie uruchamiania Amiga, albo po jej zresetowaniu, przed włączeniem kanału DMA dla Coppera trzeba zainicjalizować jeden z rejestrów lokacji Coppera (COP1LC lub COP2LC) oraz aktywować rejestru skoku. Najczęściej używa się do tego celu rejestru COP1LC, gdyć to ten rejestr jest automatycznie użyty ponownie po każdym okresie wygaszenia pionowego. Poniższa sekwencja instrukcji pokazuje jak zainicjalizować rejestr lokacji. Założono, że użytkownik stworzył poprawną copperlistę i umieścił ją pod adresem „mycoplist”.

```
;
; zainstaluj copperlistę
;
      LEA  CUSTOM,a1          ; a1 = adres bazowy układów specjalizowanych
      LEA  MYCOPLIST(pc),a0   ; adres naszej copperlisty
      MOVE.L a0,COP1LC(a1)    ; zapisz długie słowo adresu
      MOVE.W COPJMP1(a1),d0   ; spowoduje załadowanie licznika programu z COP1LC
;
; następnie włącz kanał DMA dla Coppera i rastra
;
      MOVE.W #(DMAF_SETCLR!DMAF_COPPER!DMAF_RASTER!DMAF_MASTER),DMACON(a1)
;
```

Jeśli zawartość COP1LC nie zmieni się, za każdym razem kiedy następuje pionowe wygaszenie Copper zrestartuje się z tą samą copperlistą dla każdej ramki obrazu. Spowoduje to utworzenie nieskończonej pętli, dzięki której, jeśli copperlista będzie poprawna, wygenerowany obraz będzie stabilny.

ZATRZYMANIE COPPERA

Copper nie ma na swojej liście instrukcji komendy „stop”. Aby się upewnić, że Copper nie będzie wykonywał żadnych instrukcji aż wyświetlanie ekranu się nie zakończy i licznik programu Coppera nie przeskoczy ponownie do początku copperlisty ostatnią instrukcją tej listy powinna być komenda oczekiwania na zdarzenie, które nigdy się nie pojawi. Typową instrukcją realizującą ten cel jest oczekiwanie na pozycję pionową $FF i poziomą $FE. Jak wiemy, pozycja pozioma większa niż $E2 nie jest możliwa do osiągnięcia. Kiedy wiązka kończy wyświetlanie obrazu rozpoczyna się okres wygaszania pionowego, Copper automatycznie przechodzi na początek copperlisty a warunek z ostatniej instrukcji WAIT nigdy nie zostanie spełniony.

Copper moża też zatrzymać przez wyłączenie mu dostępu do jego kanału DMA. Rejestr DMA służy do kontroli dostępu do różnych kanałów DMA. Bit 7, COPEN, włącza kanał DMA dla Coppera jeśli jest ustawiony wartością 1.

Informacje o kontroli DMA opisano w rozdziale 7 – „Układy kontroli systemu”.

Techniki zaawansowane

INSTRUKCJA SKIP

Instrukcja SKIP nakazuje Copperowi pominięcie następującej po niej instrukcji, jeśli wiązka wizyjna osiągnie lub przekroczy zadaną pozycję na ekranie.

Zawartość słów instrukcji SKIP przedstawiono poniżej. Jest niemalże identyczna jak dla instrukcji WAIT. Jedyną różnicą jest bit 0 drugiego słowa, który zawsze jest ustawiony, odróżniając tym samym instrukcję SKIP od WAIT (dla WAIT bit 0 jest zawsze wyczyszczony).

PIERWSZE SŁOWO INSTRUKCJI SKIP (IR1)

Bit 0	Zawsze włączony (1)
Bity 15 – 8	Pozycja wiązki w pionie (VP)
Bity 7 – 1	Pozycja wiązki w poziomie (HP) Pomiń jeśli licznik pozycji wiązki jest równy lub większy od wartości zapisanej na bitach 15 – 1

DRUGIE SŁOWO INSTRUKCJI SKIP (IR2)

Bity 0	Zawsze włączony (1)
Bit 15	Bit blittera. Zazwyczaj włączony (1) (Zobacz sekcję „Używanie Coppera z bitterem” poniżej)
Bity 14 – 8	Bity maski pozycji pionowej (VE)
Bity 7 – 1	Bity maski pozycji poziomej (HE)

Uwagi o poziomej i pionowej pozycji wiązki dotyczące instrukcji WAIT dotyczą rówież instrukcji SKIP.

Poniższy przykład pokazuje użycie instrukcji SKIP do pominięcia następującej po niej dowolnej instrukcji jeśli pionowa pozycja promienia wizji osiągnie lub przekroczy wartość 100 ($64).

```
DC.W $6401,$FF01  ; jeśli VP >= 100 pomiń następną instrukcję
                  ; HP jest ignorowana
```

PĘTLE I SKOKI W PROGRAMACH COPPERA ORAZ WARUNKI PORÓWNANIA

Zawartość rejestrów lokacji może byc zmieniana w dowolnym momencie. Można użyć tej techniki do twprzenia zapętleń w programach Coppera. Jednak przed kolejnym pionowym wygaszeniem COP1LC musi znów wkazywać na początek właściwej copperlisty, gdyż wartość z tego rejestru, w czasie najbliższego wygaszenia pionowego, zostanie przepisana do licznika programu Coppera.

Za określenie które bity pozycji poziomej i pionowej będą brane pod uwagę przy porównywaniu pozycji wiązki na ekranie odpowiadają bity 14 – 1 drugiego słowa instrukcji WAIT albo SKIP. Rezultat pozycji określonej przez pierwsze słowo instrukcji oraz maski bitowej z drugiego słowa jest brany pod uwagę przy porównaniu z faktycznym położeniem promienia wizyjnego zanim podjęte zostanie dalsze działanie. Odpowiedni bit pozycji brany jest pod uwagę przy porównaniu *tylko i wyłącznie jeśli* odpowiadający mu bit maski ma wartość równą 1. Jeśli bit maski ma wartość 0, rezultatem porównania dla tego bitu jest zawsze prawda (ang. *true*). Przykładowo, jeśli interesuje nas jedynie ostatnie 4 bity pozycji pionowej, w słowie maski jedynie bity (11 – 8) powinny mieć wartość 1.

Nie wszystkie bity w liczniku pozycji wiązki mogą zostać zamaskowane. Jeśli wrócimy do opisu IR2 (drugiego słowa instrukcji) zauważymy, że bit 15 jest bitem blittera. Nie służy on do maskowania pozycji wiązki, ma zupełnie inne znaczenie w przypadku instrukcji WAIT. Z tego powodu najbardziej znaczący bit pozycji promienia wizji nie może zostać zamaskowany. W większości przypadków w niczym to nie przeszkadza, jednakże poniżej opisano jak sobie radzić z takimi przypadkami.

PRZYKŁAD PĘTLI COPPERA

Zajmiemy się teraz przypadkiem, kiedy Copper zgłąsza przerwanie po każdyh 16 liniach wyrysowanych przez wiązkę. Wydawać by się mogło, że wystarczy użyć do tego celu maski $0F. Operacja porównania powinna zwrócić „true” dla każdej linii w liczniku pozycji równej $1F, $2F, $3F, itd. Biorąc pod uwagę, że warunek porównania to „równy lub większy”, wydaje się, że problem rozwiązany. Niestety, jak wspomniano wyżej, najbardziej znaczący bit pozycji zawsze jestprany pod uwagę przy porównaniu. Jeśli Copper czeka na pozycję $0F i wiązka elektronów przekroczy linię 128 ($80) rezultat porównania zawsze będzie prawdziwy. W tym przypadku minimalną wielkością porównywaną będzie $80, która zawsze jest większa od $0F i od tej pozycji przerwanie będzie zgłaszane już w każdej linii. Bardzo istotnym jest zpamiętanie, że Copper zawsze sprawdza warunek „równy lub większy”.

W poniższym przykładnie copperlisty są ułożone w pętlę. Zawartość rejestrów COP1LC i COP2LC jest ustawiona albo przez procesor albo przez Coppera gdzieś przed pętlami pokazanymi poniżej. Zakładamy również, że serwowanie przerwania Coppera co 16 linię zostało skonfigurowane gdzieś wcześniej. Przykład dotyczy widobu bez przeplotu.

Jak działa poniższy przykład? Obie pętle, w ogromnej większości, są identyczne. W każdej z nich Copper czeka aż rejestr pozycji pionowej osiągnie wartość $xF (gdzie „x” jest dowolną możliwą cyfrą szesnastkową) dla której zgłaszamy przerwanie Coppera. Aby się upewnić, że Copper nie wykona pętli ponownie zanim pionowa pozycja zmieni się (co spowodowałoby ponowne zgłoszenie przerwania jeszcze w tej samej linii), po każdym przerwaniu oczekujemy na poziomą pozycję $E2. Jak pamiętamy, pozycja ta (punkt 113) to ostatnia pozycja w linii osiągalna dla Coppera. Spowoduje to, że Copper osiągnie następną linię przed wykonaniem kolejnej instrukcji WAIT. Pętla jest startowana po zapisaniu dowolnej wartości do rejestru COPJMP1 w wynku czego Copper rozpoczyna wykonywanie instrukcji spod adresu z rejestru COP1LC.

Przez problem z maską opisany powyżej poniższy kod będzie działał błędnie po przekroczeniu pionowej pozycji 127. Trzeba uruchomić odrębną pętlę dla numerów linii równych lub większych niż 127. Kiedy pozycja zwiększy się do 127, instrukcja z pierwszej pętli jest omijana i Copper jest przerzucany do drugiej pętli. W niej następuje oczekiwanie na pozycję $xF przy założeniu, że najwyższy bit w słowie jest ustawiony (dwójkowo 1xxx1111). Warunek będzie spełniony zarówno dla pionowych i poziomych pozycji w instrukcji WAIT. Aby wystartować drugą pętlę, rejestr COPJMP2 zapisywany jest dowolną wartością. Copperlista kończy się oczekiwaniem na pozycję VP >= 255, co spowoduje zatrzymanie Coppera aż do osiągnięcia stanu wygaszenia pionowego. Pod koniec wygaszenia system operacyjny ustawia COP1LC i pierwsza pętla może zacząć się ponownie.

> *COP1LC jest ustawiany na końcu interwału wygaszania pionowego*. Układ prezerwań graficznych (serwowany przez systemową kolejkę przerwań graficznych) ustawia rejestr COP1LC pod koniec każdego wygaszania pionowego. Tak długo jak systemowy serwer przerwań będzie aktywny, COP1LC będzie zawsze ustawiany pod koniec okresu wygaszania.

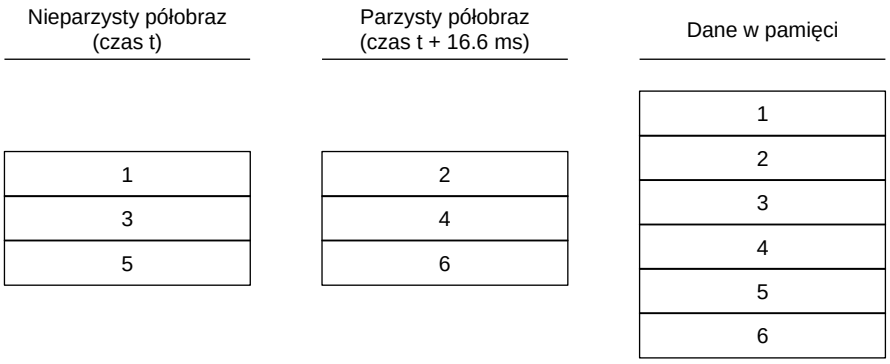
```
;
; To są dane dla copperlisty
;
; Założenia:
; adres COPPERL1 jest załadowany do rejestru COP1LC
; adres COPPERL2 jest załadowany do rejestru COP2LC
; przez jakiś inny fragment kodu.
;
COPPERL1:
      DC.W $0F01,$8F00        ; czekaj na pozycję VP=0xxx1111
      DC.W INTREQ,$8010       ; ustaw bit przerwania Coppera
      DC.W $00E3,$80FE        ; czekaj na pozycję poziomą $E2 aby się upewnić
                              ; że osiągniemy następną linię przed instrukcją
                              ; WAIT powyżej
      DC.W $7F01,$7F01        ; pomiń następną jeśli VP>=127
      DC.W COPJMP1,$0         ; uruchom ponownie Coppera od COP1LC
COPPERL2:
      DC.W $8F01,$8F00        ; czekaj na pozycję VP=1xxx1111
      DC.W INTREQ,$8010       ; ustaw bit przerwania Coppera
      DC.W $80E3,$80FE        ; czekaj na pozycję poziomą $E2 aby się upewnić
                              ; że osiągniemy następną linię przed instrukcją
                              ; WAIT powyżej
      DC.W $FF01,$FE01        ; pomiń następną jeśli VP>=255
      DC.W COPJMP2,$0         ; uruchom ponownie Coppera od COP2LC

; Ponieważ w NTSC ,mamy 262 linie a my zatrzymaliśmy się w lini 255
; Zostało nam jeszcze trochę czasu, możemy go wykorzystać na cokolwiek

      DC.W $FFFF,$FFFE        ; koniec copperlisty
;
```

UŻYWANIE COPPERA W TRYBIE Z PRZEPLOTEM

Ekran w trybie z przeplotem wyświetla dwukrotnie więcej linii niż ekran bez przeplotu. W systemie NTSC mamy więc 524 linie zamiast 262 a w systemie PAL jest tych linii 625 zamiast 312. W trybie z przeplotem wiązka wideo przechodzi przez cały ekran dwukrotnie wyświetlając w czasie jednego przejścia połowę linii (262 dla NTSC). W czasie pierwszego przejścia wyświetlane są tylko linie nieparzyste. Drugie przejście powoduje pojawienie się tylko linii parzystych na przemian do wcześniej wyświetlonych nieparzystych (stąd nazwa „przeplot”). Układ wyświetlania obrazu dzieli ekran na dwa półobrazy, jeden zawierający tylko linie parzyste a drugi tylko nieparzyste. W jaki sposób obraz z przeplotem jest przechowywany w pamięcu przedstawiono na rysunku 2-1 poniżej.



Rys. 2-1: Bitplany z trybie z przeplotem w pamięci RAM

System pobiera dane dla wyświetlanych bitplanów spod adresów wskazywanych przez odpowiednie rejestry na odpowiednie obszary pamięci. Jak widać z rysunku powyżej, adres dla klatki zawierającej parzyste linie jest o długość jeden linii większy od adresu wskazującego na dane klatki z nieparzystymi lilniami. W związku z tym wskaźnik bitplanu powinien być różny dla różnych półobrazów obrazu z przeplotem.

Krótko mówiąc, ułożenie danych w pamięci odpowiada temu jak obraz jest wyświetlany na ekanie. (parzyste i nieparzyste linie ułozone naprzemiennie). Samo wyświetlenie tych danych odbywa się dzięki dwóm oddzielnym copperlistom, każda z nich jest odpowiedzialna za własny półobraz.

Aby Copper mógł wykonać właściwą listę instrukcji, można użyć przerwania procesora 60x0 zaraz po pierwszej lini obrazu. W czasie wykonania przerwania trzeba zmienić zawatość rejestru COP1LC aby wskazywała na drugą copperlistę. W czasie wygaszania pionowego rejestr COP1LC zostanie ponownie ustawiony na pierwszą listę instrukcji.

Więcej informacji i obrazach z przeplotem można znaleźć w rozdziale trzecim opisującym playfieldy.

UŻYWANIE COPPERA Z BLITTEREM

Jeśli Copper ma być użyty do rozpoczęcia programu blittera, musi najpierw poczekać na przerwanie pojawiające się za każdym razem kiedy blitter zakończy swoje zadanie. Zmiana zawartości rejestrów blittera w czasie jego pracy może wywołać nieprzewidywalne rezultaty. W tym celu instrukcja WAIT zawiera dodatkowy bit nazwany BFD (ang. *blitter finished disable*). Standardowo bit ten ma wartość 1, wtedy tylko licznik pozycji promienia wizji ma wpływ na resultat instrukcji WAIT.

Jeśli tylko bit BFD przyjmie wartość 0, logika instrukcji WAIT ulega zmianie. Copper będzie nadal czekał na zadaną pozycję wiązki, ale dodatkowo będzie sprawdzał, czy blitter zakończył pracę. Informuje o tym flaga „koniec-pracy-blittera” (ang. *blitter-finished*). Czyszczenie tej flagi powinno być wykonywane z rozwagą. Może to spowodować, że niektóre obiekty na ekranie nie będą prawidłowo wyświetlone.

Więcej informacji znajduje się w rozdziale szóstym „Blitter”.

COPPER I 680X0

W przypadkach kiedy instrukcje Coppera są niewystarczające, można poprosić o pomoc procesor 680x0. Procesor może odpytać rejestr INTREQ o różne flagi przerwań. Aby wywołać przerwanie samego procesora, instrukcja MOVE Coppera powinna ustawić następujące bity w tym rejestrze:

Numer bitu	Nazwa	Funkcja
15	SET/CLR	Ustaw lub skasuj bit. Określa czy bity oznaczone jedynką będą ustawione czy wyczyszczone.
4	COPEN	Copper przerywa pracę 680x0

Tabela 2-1: Przerwanie pracy 680x0

Wiecej informacji o przerwaniach znajduje się w rozdziale 7, „Układy kontroli systemu”

Podsumowanie wiadomości o instrukcjach Coppera

Tabela poniżej przedstawia konfigurację bitów każdej z instrukcji Coppera. W dodatku A znajdziesz listę wszystkich rejestrów

	Move		**Wait**		**Skip**	
Numer bitu	**IR1**	**IR2**	**IR1**	**IR2**	**IR1**	**IR2**
15	X	RD15	VP7	BFD	VP7	BFD
14	X	RD14	VP6	VE6	VP6	VE6
13	X	RD13	VP5	VE5	VP5	VE5
12	X	RD12	VP4	VE4	VP4	VE4
11	X	RD11	VP3	VE3	VP3	VE3
10	X	RD10	VP2	VE2	VP2	VE2
09	X	RD09	VP1	VE1	VP1	VE1
08	DA8	RD08	VP0	VE0	VP0	VE0
07	DA7	RD07	HP8	HE8	HP8	HE8
06	DA6	RD06	HP7	HE7	HP7	HE7
05	DA5	RD05	HP6	HE6	HP6	HE6
04	DA4	RD04	HP5	HE5	HP5	HE5
03	DA3	RD03	HP4	HE4	HP4	HE4
02	DA2	RD02	HP3	HE3	HP3	HE3
01	DA1	RD01	HP2	HE2	HP2	HE2
00	0	RD00	1	0	1	1

X = nie używany, powinien mieć wartość 0 dla zgodności w przyszłości
IR1 = pierwsze słowo instrukcji
IR2 = drugie słowo instrukcji
DA = adres docelowy (ang. *dstination address*)
RD = dane w pamięci RAM, które będą zapisane w rejestrze docelowym (ang. *RAM data*)
VP = bit pionowej pozycji wiązki (ang. *Vertical Position*)
HP = bit poziomej pozycji wiązki (ang. *Horizontal Position*)
VE = bit maski dla pozycji pionowej (ang. *Vertical Enable*)
HE = mist maski dla pozycji poziomej (ang. *Horizontal Enable*)
BFD = bit wyłączający sprawdzanie statusu blittera (ang. *Blitter-Finished Disable*)

Tabela 2-2: Podsumowanie instrucji Coppera

Copper w układach ECS. Więcej informacji dotyczących Coppera w układach ECS znajduje się w dodatku C.