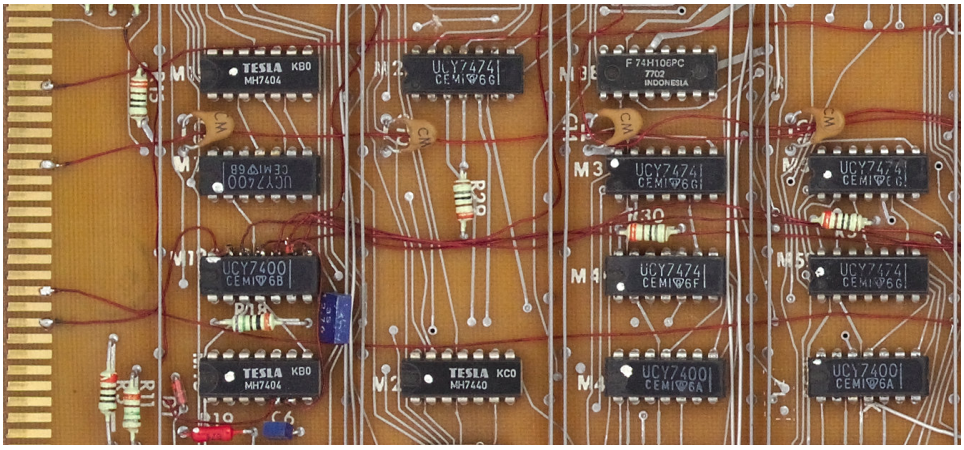
Modyfikacje procesora MERY-400

W czasach, gdy procesor kojarzony jest już wyłącznie ze złożonym, monolitycznym układem scalonym, mówienie o jego modyfikacjach może wywoływać zdziwienie. Jednak zanim pojawił się pierwszy zintegrowany wojskowy procesor F14 CADC, którego produkcję rozpoczęto w roku 1970, oraz pierwszy komercyjny układ Intel 4004, produkowany od roku 1971, jednostki centralne komputerów budowane były z elementów dyskretnych i podstawowych układów scalonych w technologiach DTL lub TTL.

Datowany na koniec lat '70 procesor MERY-400 ma właśnie taką konstrukcję. Jest sporym modułem, składającym się z dziewięciu pakietów – płyt o wymiarach 30x30cm. Pakiety zbudowane są niemal wyłącznie w oparciu o układy scalone TTL z serii 7400 oraz elementy pasywne. Taka konstrukcja powoduje, że z lutownicą w dłoni można procesor w zasadzie dowolnie przerabiać, a zadanie ułatwia dodatkowo fakt, że na poszczególnych pakietach można znaleźć niewykorzystane bramki i przerzutniki.



Fot. 1: Modyfikacje na pakiecie przerwań procesora MERY-400

W latach '80, w Instytucie Okrętowym Politechniki Gdańskiej, w zespole rozwijającym system operacyjny CROOK, oraz w Amepolu, który od roku 1984 kontynuował rozwój MERY-400, powstało kilka modyfikacji procesora MERY-400. Wykorzystywał je zarówno CROOK jak i oprogramowanie na nim działające. Były one najpierw implementowane w istniejących instalacjach MERY-400, a ostatecznie wszystkie złożyły się na poprawioną wersję procesora używanego w komputerze MX-16.

Nie zachowała się do dziś żadna dokumentacja tych modyfikacji. Niniejsze opracowanie jest wynikiem analizy zmian w budowie procesora MERY-400, na której pracowali autorzy CROOK-a, wprowadzonych względem oryginalnej konstrukcji. Komputer ten pracował do roku 1991, a jego procesor zawierał wszystkie istniejące wtedy zmiany.

Przeróbki procesora dotyczą pakietów: arytmometru (P-A), dekodera (P-D), rejestrów (P-R) i przerwań (P-P). Ich umiejscowienie niekoniecznie związane jest z funkcjonalnością pakietu, a raczej z obecnością wolnych bramek i przerzutników. Schematy przedstawione dalej pokazują zmiany (wyróżnione kolorem czerwonym) względem schematów znajdujących się w dokumentacji

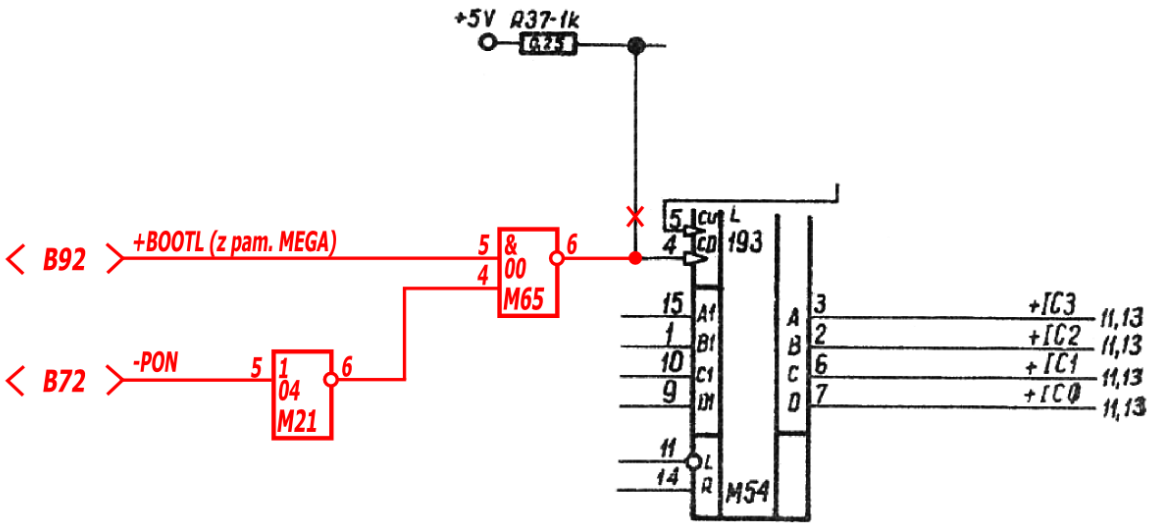
techniczno-ruchowej. Nazwy nowych sygnałów, nie istniejących w dokumentacji, zostały nadane w celu ułatwienia przedstawienia zmian.

Automatyczny bootstrap

Pamięć MEGA Amepolu zawiera bootloader systemu operacyjnego zapisany w pamięci EPROM, dostępnej zaraz po uruchomieniu maszyny od adresu *0xf000* w bloku zerowym. Można uruchomić zawarty tam kod ustawiając z pulpitu technicznego odpowiednią zawartość licznika rozkazów i startując procesor. Problem pojawia się w przypadku komputera MX-16, który dostarczany był bez pulpitu technicznego – bootloader musiał w jakiś sposób zostać załadowany automatycznie.

Zostało to zrobione poprzez modyfikację procesora, która po włączeniu zasilania powoduje ustawienie licznika rozkazów na adres *0xf000*. Przeróbka ta nie ma żadnego efektu, jeśli w systemie nie jest zainstalowana pamięć MEGA, bądź jeśli bootstrap z pamięci MEGA nie jest możliwy.

Do pakietu P-A doprowadzony jest sygnał *+BOOTL* z pamięci MEGA. Jego dokładne źródło nie jest znane ze względu na brak schematów pamięci MEGA, jednak charakter modyfikacji wskazuje, że informuje on o gotowości pamięci do dostarczenia programu rozruchowego od adresu *0xf000*. Sygnał *-PON* generowany jest w zasilaczu i informuje o ustabilizowaniu napięć zasilających.



Rys. 2: Zmiany w schemacie pakietu P-A3-2, arkusz 10

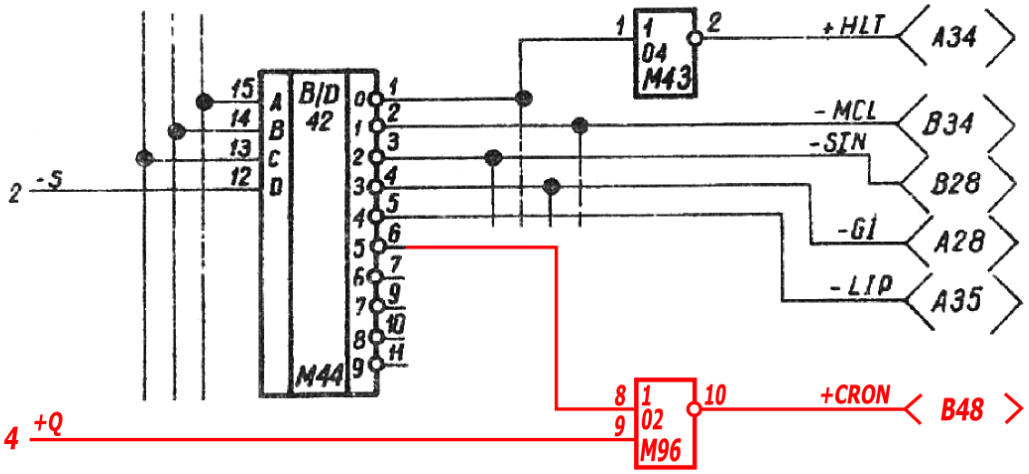
Układ M54 to 4-bit rejestr przechowujący najstarszy kwartet bitów licznika rozkazów IC. Istniejące połączenie jego wejścia *CD* (*Count Down*) do stanu wysokiego zostało przerwane i zastąpione iloczynem sygnałów *+BOOTL* i *+PON*. Aktywne *0* na wyjściu bramki NAND (układ M21) powoduje zmniejszenie o jeden zawartości rejestru M54, a co za tym idzie ustawienie licznika rozkazów na adres *0xf000*.

Instrukcja włączająca modyfikacje

Ponieważ konieczne było zapewnienie możliwie najpełniejszej kompatybilności wstecznej z oryginalnym procesorem MERY-400, funkcjonalność opisanych dalej przeróbek dostępna jest dopiero po programowym ich aktywowaniu nową instrukcją procesora.

Instrukcję taką, nazwaną CRON, o kodzie *0166500* (ósemkowo), dodano w grupie rozkazów S. Maszyna zmodyfikowana uruchamia się domyślnie w trybie kompatybilności z oryginalnym procesorem, a dopiero wywołanie rozkazu CRON aktywuje modyfikacje. Zerowanie maszyny kluczem CLEAR bądź rozkazem MCL przywraca procesor do pierwotnego stanu kompatybilności.

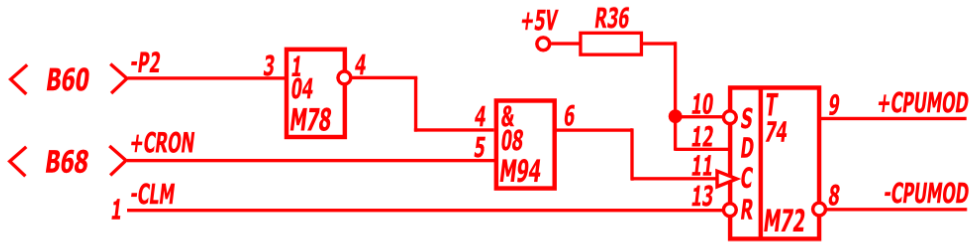
Modyfikacja obejmuje pakiety P-D i P-P. Na pakiecie P-D użyto wyjścia 6 układu M44, na którym stan niski informuje o wartości *101* (binarnie) w polu A rozkazu z grupy S. Sygnał ten jest rozpatrywany w iloczynie z niskim poziomem sygnału *+Q*. Tak wypracowany sygnał *+CRON* wskazuje na wydanie przez system operacyjny instrukcji CRON.



Rys. 3: Zmiany w schemacie pakietu P-D3-2, arkusz 3

Należy zauważyć, że instrukcja ta wciąż jest instrukcją nielegalną, co zapewnia kompatybilność wsteczną. Programista systemu operacyjnego, używający instrukcji CRON, musi obsłużyć fakt, że po jej wydaniu zgłoszone zostanie przerwanie „niepoprawny rozkaz”.

Rezultatem wydania instrukcji CRON musi być przełączenie procesora w tryb pracy z modyfikacjami aż do następnego zerowania, a więc sygnał *+CRON* musi zostać gdzieś zapamiętany. Dzieje się to na pakiecie P-P.



Rys. 4: Dodatek do schematu pakietu P-P3-2

Przerzutnik M72 służy do zapamiętania faktu aktywowania modyfikacji. Na wejście D przerzutnika podana jest na stałe wartość *1*, a wejściem sterującym, powodującym jej wpisanie, jest wejście zegarowe. Podany jest na nie sygnał *+CRON* w iloczynie z sygnałem *+P2*, który informuje, że procesor jest w stanie P2 – wykryta została instrukcja nieefektywna (a taką instrukcja CRON jest).

Jedynym sposobem na wyzerowanie wartości zapisanej w przerzutniku jest podanie niskiego poziomu na wejście R. Doprowadzony do niego jest sygnał *-CLM* sygnalizujący zerowanie maszyny. Kolejne powtórzenia wywołań instrukcji CRON nie wnoszą niczego – do przerzutnika wpisywana jest ponownie *1*.

17-bitowe adresowanie bajtów

W oryginalnej konstrukcji procesora adres bajtu jest – podobnie jak adres słowa – 16-bitowy. 15 najstarszych bitów adresuje słowo w pamięci, a najmłodszy bit wskazuje lewy (*0*) lub prawy (*1*) bajt w 16-bitowym słowie maszyny. Oznacza to, że za pomocą instrukcji używających adresowania bajtowego: LB, RB, CB można odwołać się jedynie do pierwszych 2^{15} słów w 64k segmencie pamięci, czyli do połowy jego przestrzeni. Fakt ten jest zresztą źródłem nieścisłości w DTR MERY-400.

Adresowanie bajtu w nie zmodyfikowanym procesorze realizowane jest w następujący sposób:

- Rozkaz zawiera 16-bit adres bajtu w postaci: 15 bitów adresu słowa, 1 bit (najmłodszy) wskazujący bajt w słowie.
- Tak dostarczony adres procesor przesuwa o jeden bit w prawo. W wyniku otrzymuje 16-bitowy adres słowa w pamięci, którego najstarszy bit jest zawsze zerem.
- „Wypadający” najmłodszy bit zachowywany jest, aby później na podstawie jego wartości z komórki pamięci wybrać odpowiedni bajt (lewy bądź prawy).

Adres bajtu, jako argument normalny, podlega również pre- i B-modyfikacji. Obie te operacje są niczym innym, jak dodawaniem modyfikatorów do argumentu, wykonywanym w arytmometrze procesora, w ramach mikrooperacji będących składowymi rozkazu używającego adresowania bajtowego. W wyniku takiego dodawania może więc wystąpić przeniesienie, które w standardowym procesorze jest zaniedbywane.

Przeróbka procesora zmienia to zachowanie. Gdy do pierwotnego argumentu dodawana jest pre-, lub B-modyfikacja, przeniesienie, które ostatecznie występuje przy tej operacji zapamiętywane jest

w dodatkowym rejestrze. Później, kiedy adres bajtowy przesuwany jest w prawo, aby otrzymać docelowy adres słowa w pamięci, na najstarszy bit adresu wsuwane jest nie zero, a zawartość ostatniego zapamiętanego przeniesienia. W ten sposób, używając pre- lub B-modyfikacji można przygotować 17-bitowy adres bajtu, dzięki czemu możliwe staje się zaadresowanie dowolnego bajtu w całym 64k słowowym bloku.

Poniższy przykład używa B-modyfikacji do skonstruowania 17-bit adresu bajtu (składnia assemblera EMAS).

```
LW r2, 0x9400    (1)
MD 1             (2)
LB r1, r2+r2     (3)
```

Do rejestru *r2* ładowana jest wartość *0x9400* – adres słowa, w którym adresowany będzie bajt (1). Instrukcja pre-modyfikacji MD (2) powoduje, że w następnej instrukcji, do argumentu efektywnego dodawana będzie wartość *1*, wskazująca prawy bajt w słowie. Następna instrukcja (3) ładuje do rejestru *r1* prawy bajt słowa o adresie *0x9400* używając B-modyfikowanego argumentu (*r2+r2*), dodatkowo pre-modyfikowanego wspomnianą wartością *1*. W trakcie obliczania argumentu efektywnego ostatniej instrukcji wystąpi przeniesienie, które zostanie zapamiętane:

0x9400 + 0x9400 + 1 = 0x12801

Po przesunięciu wyniku o jedną pozycję w prawo z uwzględnieniem przeniesienia, procesor otrzyma poprawny adres słowa (*0x9400*), oraz „wypadającą” jedynkę, która wskaże prawy bajt. W procesorze bez przeróbki, bit przeniesienia zostałby stracony, a ostatecznym adresem słowa stałaby się niewłaściwa wartość *0x1400*.

Następny przykład używa do wypracowania adresu bajtowego pre-modyfikacji.

```
LW r5, 0xAAAA    (1)
LWT r2, 1        (2)
...
LW r6, r5+r1     (3)
MD r6+r2         (4)
RB r3, r6        (5)
```

W rejestrze *r5* przechowywany jest adres bazowy obszaru pamięci, *0xAAAA* (1). Załóżmy, że w rejestrze *r1* przechowywane jest pewne przesunięcie względem adresu bazowego (co ma praktyczne zastosowanie w pętlach programowych). Rejestr *r2* zawiera wartość wskazującą bajt w adresowanym słowie (2). Do rejestru *r6* ładowany jest adres bazowy z przesunięciem (3). Modyfikacja (4) dodaje do niego zawartość rejestru *r2*, która wskazuje bajt w słowie. Rozkaz pamiętania bajtu (5) używa jako argumentu normalnego zawartości r6, która pre-modyfikowana zostanie poprzednią instrukcją. W rezultacie otrzymamy efektywny adres bajtowy, którego najstarszy bit zostanie zapamiętany jako przeniesienie i użyty będzie do wyprowadzenia adresu słowa:

*(r5 + r1) + r2 +(r5 + r1) = 2 * 0xAAAA + 2 * r1 + r2 = 0x15554 + 2 * r1 + r2*

Tak zrealizowane 17-bitowe adresowanie bajtów ma jednak wadę. W oprogramowaniu napisanym na oryginalny, nie zmodyfikowany procesor, może zostać użyta konstrukcja z pre- lub B-modyfikacją, która na procesorze przerobionym przyniesie nieoczekiwany skutek. Rozpatrzmy następujący przykład:

```
LW r2, 0x3b00
LWT r3, -12
MD r2
LB r1, r2+r3
```

Jeśli przeniesienie przy obliczaniu argumentu normalnego jest zaniedbywane, jak to ma miejsce w oryginalnym procesorze, adres bajtowy zostanie w ostatniej instrukcji obliczony zgodnie z intencją programisty w następujący sposób:

1. Argument pierwotny w *r2* ma wartość *0x3b00*.
2. Dodanie pre-modyfikacji o wartości *r2=0x3b00* da *0x7600* (bez przeniesienia).
3. Dodanie B-modyfikacji (*r3=-12*) da *0x75f4* i zapalona zostanie flaga przeniesienia.
4. Przesunięcie wyniku w prawo dla otrzymania adresu bajtowego da wartość *0x3afa.*

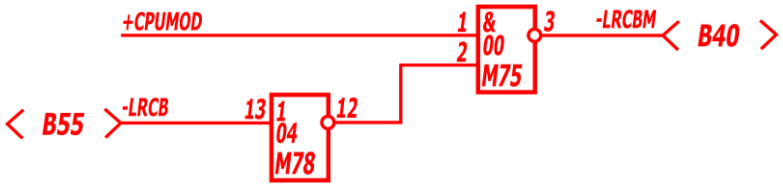
W kroku 3 występuje przeniesienie, ponieważ arytmometr do argumentu dodaje B-modyfikację, która w zapisie U2 ma wartość *0xfff4*. Jeśli procesor jest zmodyfikowany, to zapamiętane przeniesienie zostanie w ostatnim kroku użyte i wsunięte na najstarszą pozycję ostatecznego wyniku, dając w efekcie ostateczny adres słowa *0xbafa*, zamiast spodziewanego *0x3afa*.

W związku z powyższym należało więc zapewnić możliwość poprawnego uruchamiania na zmodyfikowanym procesorze, pod kontrolą systemu operacyjnego, który aktywował modyfikacje, również takich programów. Zostało to osiągnięte przez ustalenie następujących alternatywnych warunków, w których 17-bit adresowanie bajtów jest używane:

- wykonywany jest program w trybie systemu operacyjnego (*Q=0*), albo
- wykonywany jest program w trybie użytkownika i nie jest ustawiony bit specjalny (*BS=0*).

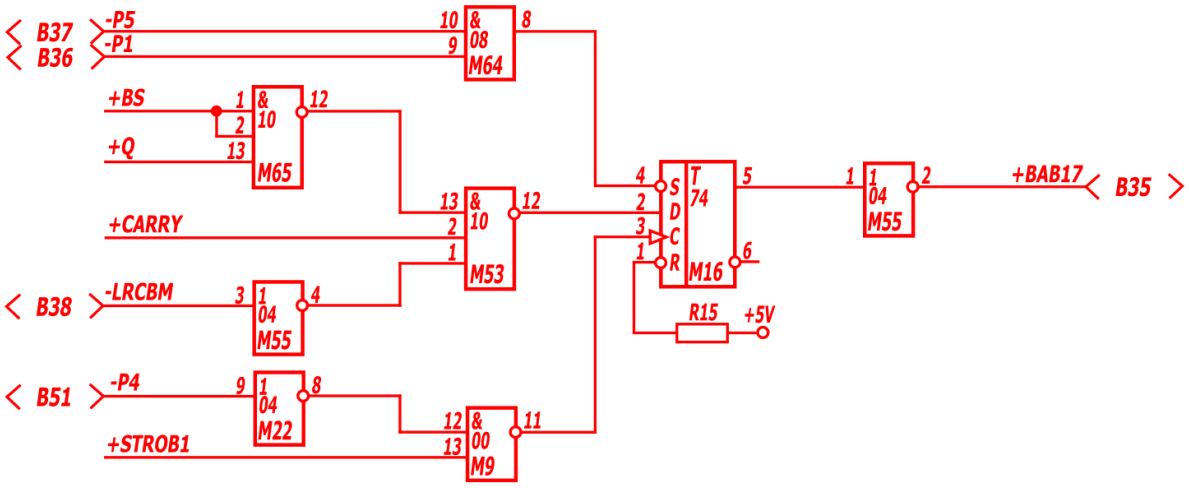
Pierwsza część warunku jest oczywista – jeśli system operacyjny jest przeznaczony dla zmodyfikowanego procesora, to musi używać poprawnego adresowania bajtów. Druga część warunku wykorzystuje bit specjalny BS, który w założeniu twórców procesora nie odgrywał żadnej roli dla programów działających w blokach pamięci użytkownika. Służył jedynie systemowi operacyjnemu działającemu w dwuprocesorowej konfiguracji maszyny, umożliwiając dostęp do pamięci sąsiedniego procesora. Nie zmienia to faktu, że system operacyjny może jego wartość dowolnie ustawić przed przełączeniem kontekstu na proces użytkownika. Dzięki temu, dla programów (procesów), o których wiadomo było, że nie pracują poprawnie z 17-bit adresowaniem, system operacyjny mógł je dezaktywować, ustawiając przed przełączeniem kontekstu bit BS w rejestrze stanu SR na *1*. Takie zachowanie pozostaje jednocześnie kompatybilne z oryginalnym procesorem.

Modyfikacja ta jest jedną z najbardziej złożonych i obejmuje zmiany na pakietach P-P, P-R i P-A. Na pakiecie P-P wypracowywany jest sygnał *-LRCBM* mówiący o tym, że instrukcja LB, RB lub CB została wydana na procesorze z aktywnymi modyfikacjami.



Rys. 5: Dodatek do schematu pakietu P-P3-2

Na pakiecie P-R znajduje się przerzutnik M16, w którym pamiętany jest 17. bit używany w adresowaniu bajtowym.



Rys. 6: Dodatek do schematu pakietu P-R3-2

Przerzutnik taktowany jest sygnałem *+STROB1* w iloczynie z sygnałem *+P4*, mówiącym o tym, że procesor jest w stanie P4, w którym następuje pre- i B-modyfikacja argumentu normalnego. Na wejściu D pojawia się sygnał, który w rezultacie wpisuje do przerzutnika wartość *-CARRY*, gdy spełnione są wszystkie niezbędne warunki:

$$D = \sim(CARRY \;\&\; LRCBM \;\&\; \sim(Q \;\&\; BS))$$

czyli:

- wydany został rozkaz LB, RB lub CB na procesorze z aktywną modyfikacją, oraz
- nie jest prawdą, że procesor pracuje w trybie użytkownika (*Q=1*) i bit specjalny jest ustawiony (*BS=1*)

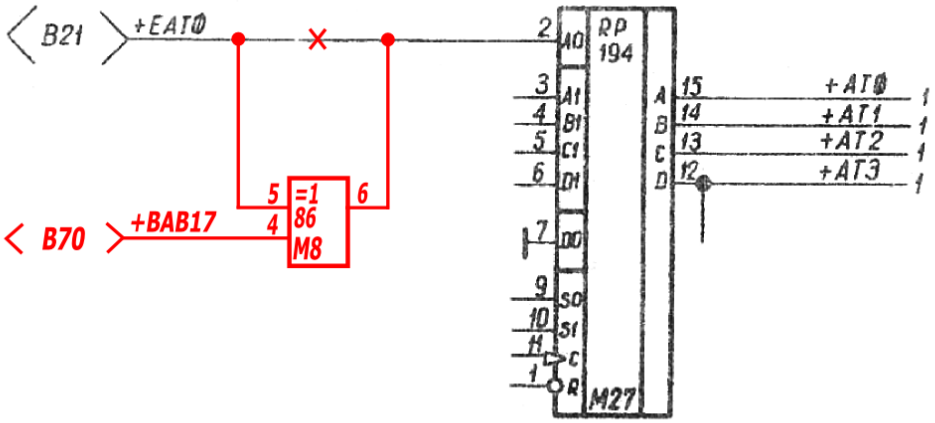
Zapamiętywana w przerzutniku wartość jest zanegowana, stąd bramka NOT (M55) na wyjściu.

Na wejście S (które w związku z negacją na wyjściu M53 jest efektywnie wejściem zerującym) podane są sygnały *-P1* (stan, w którym odczytywany jest rozkaz) i *-P5* (stan, w którym pobierany jest argument pośredni). Wystąpienie któregokolwiek z nich powoduje zapamiętanie *1*, czyli efektywne wyzerowanie zawartości przerzutnika.

Ostatecznie sygnał *+BAB17* niesie informację o 17. bicie powstałym w wyniku B-modyfikacji lub pre-modyfikacji argumentu normalnego rozkazów bajtowych:

BAB17 = CARRY & LRCBM & (~Q | ~BS)

Dzięki modyfikacji wprowadzonej na pakiecie P-A, wartość ta wsuwana jest z lewej strony do rejestru AT w chwili przesuwania jego zawartości podczas przygotowywania adresu słowa przy adresowaniu bajtowym:



Rys. 7: Zmiany w schemacie pakietu P-A3-2, arkusz 7

Przerwanie programowe o wysokim priorytecie

System operacyjny CROOK wymaga istnienia sposobu na wywoływanie niektórych wewnętrznych procedur z jednoczesnym zapewnieniem, że będą one realizowane atomowo – nie zostaną przerwane przez nadchodzące przerwania z urządzeń zewnętrznych czy zegara systemowego. Procesor MERY-400 nie daje wprost takiej możliwości. Wersje systemu CROOK od 3 do 5 rozwiązywały ten problem używając pseudoinstrukcji SIN o kodzie *036000 (ósemkowo)*. Z punktu widzenia procesora jest to instrukcja niepoprawna, generującą przerwanie „nieprawidłowy rozkaz”, znajdujące się bardzo wysoko na liście priorytetów przerwań. Procedura obsługi tego przerwania skonstruowana jest tak, aby w tym szczególnym przypadku niepoprawnego rozkazu zapewnić jego obsługę jak przerwania programowego o wysokim priorytecie, traktując kolejne słowo za nim jako adres procedury, którą należy wykonać. To obejście wykorzystywane było aż do ostatniej rewizji CROOK-5.

Istnieje jednak również alternatywne jądro CROOK-5 w rewizjach 7 i 8, potrafiące pracować z procesorem, w którym za pomocą przeróbki dodane dwie nowe, legalne instrukcje: SINT i SIND. Zastępują one funkcjonalnie pseudoinstrukcję SIN i pozwalają na atomowe wywoływanie procedur.

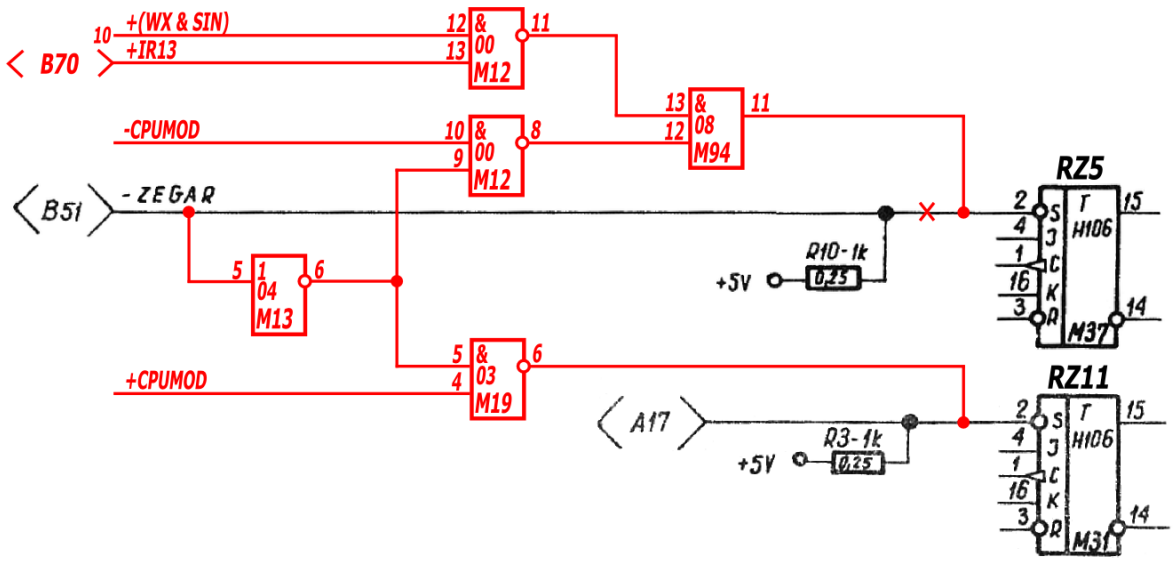
Na tę przeróbkę procesora składają się zmiany w kilku jego obszarach:

1. W dekoderze rozkazów znaleziono „miejsce” na dwie nowe instrukcje: w grupie rozkazów bezargumentowych S. Jest to jedyna grupa w obszarze rozkazów legalnych, gdzie część bitów kodu rozkazu nie jest w ogóle wykorzystana (odpowiednie linie dekodera rozkazów nie są użyte). Nowym rozkazom zostały nadane kody (ósemkowo):

- SINT = *0166204*
- SIND = *0167204*

2. Przerwanie zegarowe (pozycja 5) zostało przeniesione na pozycję 11, w standardowym procesorze nie skojarzoną z żadnym źródłem przerwania.
3. Przerwanie 5 zostało użyte jako przerwanie programowe o wysokim priorytecie, sterowane instrukcjami SINT i SIND.

Modyfikacja ma miejsce na pakiecie P-P. Oryginalne połączenie źródła przerwania zegarowego do przerzutnika M37 zostało przerwane. Przerwanie 11 nie ma w oryginalnym procesorze żadnego źródła, na wejściu S jest zawsze logiczna *1*. Zmodyfikowany układ przerwań kieruje przerwanie zegarowe na przerzutnik M31 lub M37 w zależności od tego, czy modyfikacja procesora jest (odpowiednio) aktywna, czy nie.



Rys. 8: Zmiany w schemacie pakietu P-P3-2, arkusze 4 i 5

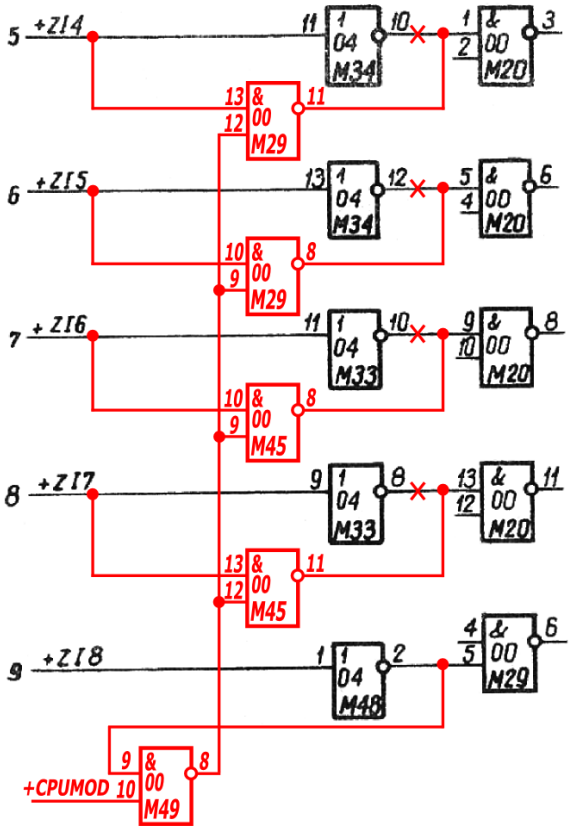
Dodatkowo badana jest pozycja 13. rejestru rozkazu (*+IR13*) w iloczynie z sygnałem *+(WX & SIN)*, oznaczającym mikrooperację WX dla grupy rozkazów SIN, SIT, SIL, CIT. Funkcjonalnie jest to dekoder rozkazów SINT i SIND, które ulokowane są w grupie ze wspomnianymi wcześniej rozkazami (rozkazy grupy S z polem *A=010*). Jeśli najstarszy bit pola C jest dla nich zapalony (*IR13=1*), to procesor zgłasza przerwanie 11.

Przeróbka ma tę drobną wadę, że jeśli modyfikacje procesora nie są aktywne, a wydana zostanie instrukcja SIND lub SINT, to zgłoszone zostanie nadmiarowe przerwanie zegarowe.

Należy też zauważyć, że instrukcje SINT i SIND nie są tak naprawdę osobnymi instrukcjami, a specjalnymi przypadkami instrukcji SIN, SIL, SIT, CIT, dla których ustawiony jest najstarszy bit pola C (ignorowany w niezmodyfikowanym procesorze). W efekcie, wydanie instrukcji SINT lub SIND, oprócz zamierzonego efektu (zgłoszenia przerwania 11), będzie miało efekt dodatkowy, zależny od zawartości dwóch najmłodszych bitów pola C. Zgodnie z treścią rozkazów SIT, SIL, SIN, CIT, które te pola wykorzystują, zgłoszone lub wyzerowane zostaną przerwania 30 i/lub 31. W systemie CROOK instrukcje SINT i SIND mają pole rozkazu *C=100*, a więc zgłoszeniu przerwania 11 towarzyszy wyzerowanie przerwań 30 i 31.

Zmiana maskowania przerwań

CROOK-5 ma jeszcze jedno wymaganie, które ostatecznie zostało spełnione za pomocą modyfikacji sprzętowej. Należało zapewnić, aby w trakcie obsługi przerwania kanałowego nie pojawiło się przerwanie o wyższym priorytecie (wynika to z tego, jak system przerwań działał w komputerze K-202, dla którego CROOK był pierwotnie pisany). W procesorze MERY-400 bez przeróbek wymuszane to było przez programowe maskowanie przerwań na początku procedury obsługi przerwań kanałowych. W procesorze przerobionym zrealizowane to było sprzętowo: Przyjęcie przerwania kanałowego powodowało automatyczne zamaskowanie przerwań od 5 do 31.



Rys. 9: Zmiany w schemacie pakietu P-P3-2, arkusze 1 i 2

Modyfikacja została wprowadzona na pakiecie P-P. W niezmodyfikowanym procesorze, na podstawie przyjętych przerwań, sygnały *+ZI4* do *+ZI7* maskują odpowiednie grupy przerwań. Jeśli modyfikacja procesora jest aktywna, sygnał *+ZI8*, oprócz swojej pierwotnej funkcji, powoduje też zamaskowanie przerwań w grupach 4, 5, 6 i 7.

Reakcja procesora na brak pamięci

Na istnienie poniższej zmiany silnie wskazuje analiza zawartości kości EPROM pamięci MEGA oraz inne zmiany w procesorze MX-16, jednak nie została ona definitywnie potwierdzona.

Ponieważ MX-16 nie był wyposażony w pulpit techniczny, konieczna była zmiana w sposobie,

w jaki procesor reaguje na odwołanie do nieskonfigurowanego segmentu pamięci w obszarze systemowym. Oryginalny procesor przechodził w stan STOP, z którego można go było wybudzić jedynie kluczem START/STOP. W MX-16, ze względu na brak pulpitu technicznego, nie było takiej możliwości, więc obsługa tego przypadku musiała być w pełni programowa i ograniczać się do zgłoszenia przerwania, podobnie jak w przypadku błędów dostępu do pamięci w segmentach programów użytkowych.

Zostało to osiągnięte nie tyle przez modyfikację, co rekonfigurację procesora. Na pakiecie P-X istnieje zwora S-R (nie opisana w dokumentacji), która decyduje o tym, czy w opisanym wyżej przypadku wymuszany jest stan STOP. W komputerach MX-16 była ona najprawdopodobniej rozwarta.

Połączenia na platerze procesora

Poza zmianami wprowadzonymi na samych pakietach, powstały też nowe połączenia na platerze procesora, niezbędne do poprowadzenia odpowiednich sygnałów między pakietami:

Z		**Do**		**Sygnał**
Pakiet	**Piórko**	**Pakiet**	**Piórko**	
P-D	B48	P-P	B68	*+CRON*
P-P	B40	P-R	B38	*-LRCBM*
P-P	B55	P-D	A45	*-LRCB*
P-P	B60	P-M	A07	*-P2*
P-P	B68	P-D	B48	*+CRON*
P-P	B70	P-M	B52	*+IR13*
P-R	B35	P-A	B70	*+BAB17*
P-R	B36	P-M	A29	*-P1*
P-R	B37	P-M	B55	*-P5*
P-R	B38	P-P	B40	*-LRCBM*
P-R	B51	P-D	A77	*-P4*
P-A	B70	P-R	B35	*+BAB17*
P-A	B72	P-M	A12	*-PON*
P-A	B92	ME-GA-400	W48	*+BOOTL*

Jakub Filipowicz, http://mera400.pl

Wrocław, marzec 2016