Emulator
elektronicznej maszyny cyfrowej
ODRA 1003 / ODRA 1013

2024-11-03

Klemens Czajka

Spis treści

O projekcie

Elektroniczna maszyna cyfrowa ODRA 1013 to komputer zbudowany w Zakładach ELWRO. Znaczący udział w jego zaprojektowaniu miał Thanasis Kamburelis (gr. Θανάσης Καμπουρέλης), „wówczas zgodnie uznawany za najlepszego w Polsce specjalistę w zakresie architektury i struktury logicznej komputerów cyfrowych” (https://pl.wikipedia.org/wiki/Thanasis_Kamburelis), główny projektant następnych polskich komputerów.

https://pl.wikipedia.org/wiki/Odra_1003
https://pl.wikipedia.org/wiki/Odra_1013

Na niej w latach 1966-1972 uczono programowania na Uniwersytecie Marii Curie-Skłodowskiej w Lublinie. Maszyna UMCS była rozbudowana o obsługę czytników i perforatorów taśmy papierowej 8-kanałowej, oraz inne modyfikacje. Nauczycielami byli m. in. Światomir Ząbek i Zbigniew Skorzyński.

Pomysł napisania emulatora tej maszyny zrodził się pod wpływem przeczytanej w internecie czyjejś nadziei na tego rodzaju program. Za radą synów, którzy wspierają mnie i kibicują w tej pracy, oparcie się na języku JavaScript i HTML ułatwiło opracowanie wizualnej strony emulatora i jego działanie na praktycznie dowolnym komputerze, z dowolnym systemem operacyjnym, byleby z jakąś w miarę współczesną przeglądarką internetową. Ceną wyboru jest mało przejrzyste oprogramowanie operacji (nawet elementarnych) na słowach 40-bitowych, wynikające ze specyfiki arytmetyki języka JavaScript.

Podstawowym pytaniem jest: do czego potrzebny jest emulator ODRY 1013? To nie jest tak, że zalegają gdzieś programy na tę maszynę i nie ma ich na czym uruchomić. Nie ma ani programów, ani maszyny – ówczesna technika to już przeszłość. Cóż, emulator ten powstaje dla własnej i innych przyjemności – takie hobby, ale nie tylko. Skoro ODRA 1003 i ODRA 1013 spełniły istotną rolę w rozwoju przemysłu komputerowego w Polsce, to emulator może je upamiętnić w postaci działającego modelu; a jako że nie ma już programów dla których jest niezbędna ODRA 1013, to ma to być model demonstracyjny, edukacyjny. Stąd pomysł, by dało się wykonywać programy z prędkością znacznie przewyższającą prędkość rzeczywistej maszyny, ale też obserwować pracę w zwolnionym tempie. Emulator ma umożliwić poznawanie ODRY i pozostać jako swoisty zabytek przechowujący pamięć o tej maszynie.

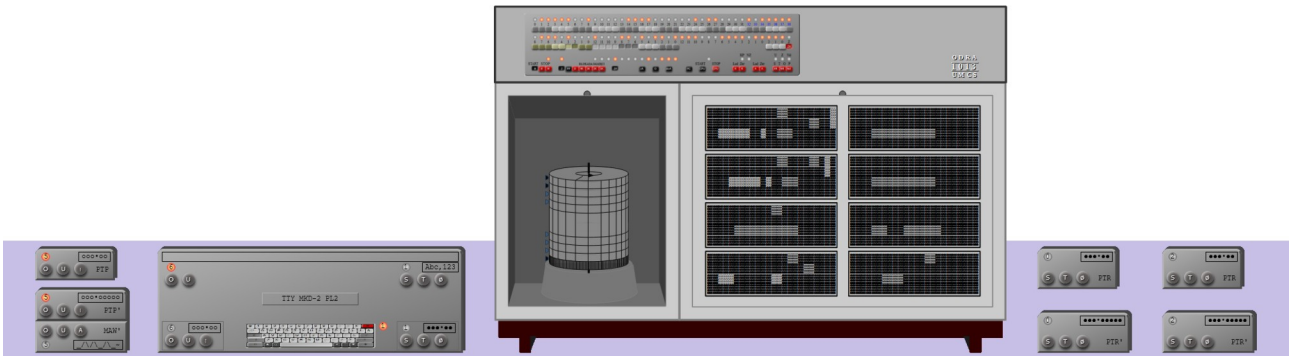
Szczegółowe informacje dotyczące ODRY 1003 i ODRY 1013 pochodzą z:

[1] artykułu „Thanasis Kamburelis, Maszyna cyfrowa ODRA 1003” zamieszczonego w „Zastosowaniach Matematyki VIII (1965)”,

[2] skryptu akademickiego UMCS „Światomir Ząbek: Programowanie i obsługa maszyn cyfrowych ODRA 1003 i ODRA 1013, Wydanie II, LUBLIN 1974”,

[3] skryptu Politechniki Wrocławskiej „Czesław Daniłowicz: Programowanie maszyn cyfrowych ODRA 1003 i ODRA 1013, WROCŁAW 1971”.

O maszynie



Informacje techniczne o ODRZE 1003 i ODRZE 1013 można znaleźć w internecie.
Zachowane egzemplarze znajdują się w Muzeum Techniki w Warszawie.

ODRA 1003

Tranzystorowy, szeregowy komputer drugiej generacji skonstruowany w 1963 roku i produkowany w Zakładach Elektronicznych ELWRO od 1964 roku. Komputer przeznaczony był do obliczeń naukowo-technicznych i sterowania procesami technologicznymi.

Komputer 1+1 adresowy, tzn. rozkaz zawiera jeden adres argumentu (lub sam argument), oraz drugi adres, wskazujący następny rozkaz do wykonania.

Arytmetyka binarna, kod uzupełnień do dwóch.

Długość słowa maszynowego: 39 bitów + 1 bit techniczny (łącznie 40 bitów)

Pamięć operacyjna: bęben magnetyczny o pojemności 8192 słów (64 ścieżki po 128 słów każda)

Urządzenia wejściowe:

- fotoelektryczny czytnik taśmy FC-1 firmy ELWRO
- dalekopis firmy Lorenz
- przystawka z konwerterem analogowo-cyfrowym firmy ELWRO

Urządzenia wyjściowe:

- perforator taśmy firmy Facit
- dalekopis firmy Lorenz

Mimo podstawowego przeznaczenia komputera, jakim były obliczenia naukowe, w roku 1962 inżynier Witold Podgórski, pracujący we wrocławskich Zakładach Elektronicznych Elwro stworzył na prototypie Odry 1003 grę logiczną Marienbad, po przeczytaniu w czasopiśmie Przekrój opisu tak właśnie nazwanej wersji gry logicznej Nim. Podgórski zaprogramował grę dla maksymalnie ośmiu tysięcy rządków zapałek, po prawie bilion zapałek w każdym; przy takim ustawieniu odpowiedź Odry na ruch gracza zajmowała niecałą godzinę. Odry nie dało się pokonać, nawet przy standardowym ustawieniu dla szesnastu zapałek. Algorytm Marienbadu przekazano Wojskowej Akademii Technicznej, gdzie chętni mogli grać z komputerem sterowanym przez operatora, przy czym władze uczelni niechętnie patrzyły na wykorzystywanie Odry do rozrywki [Bartłomiej Kluska, Bartosz Rozwadowski: Bajty polskie. Sosnowiec: 2014, s. 5-7. ISBN 978-83-927229-2-2].

ODRA 1013

Komputer skonstruowany jako rozwinięcie ODRY 1003 i produkowany w ELWRO od 1966 roku.

ODRA 1013 jako pamięć operacyjną miała, oprócz pamięci bębnowej, pamięć ferrytową o pojemności 256 słów, oraz 4 dodatkowe rozkazy szybkiego przesyłania blokowego zawartości ścieżek bębnowych na ferrytowe i odwrotnie.

W pamięci ferrytowej bit techniczny słowa pełnił rolę bitu parzystości służącego do wykrywania przekłamań zapisu w pamięci. Przekłamanie sygnalizowane było zatrzymaniem maszyny i zapaleniem lampki KP.

ODRA 1013 UMCS

Jest to ODRA 1013 zmodyfikowana w Zakładzie Metod Numerycznych UMCS. Oprócz samego komputera, były tam:

- dwa czytniki taśmy perforowanej 5-kanałowej (drugi czytnik w miejsce nieużywanej przystawki z konwerterem analogowo-cyfrowym służącej do podłączania specjalistycznych urządzeń przemysłowych),
- jeden perforator taśmy 5-kanałowej,
- dwa czytniki taśmy perforowanej 8-kanałowej, podłączane w miejsce czytników 5-kanałowych i sterowane odpowiednio dodatkowymi rozkazami 066 i 266,
- jeden perforator taśmy 8-kanałowej, podłączany w miejsce perforatora 5-kanałowego i sterowany dodatkowym rozkazem 566,
- samopis MAW do rysowania wykresów, podłączony (poprzez konwerter analogowo-cyfrowy) równolegle z perforatorem taśmy 8-kanałowej,
- i oczywiście dalekopis.

Ponadto dodano rozkaz o kodzie 446 – skok przy braku gotowości urządzenia wejściowego.

Podłączanie i przestawianie czytników i perforatorów 5-kanałowych na 8-kanałowe i odwrotnie było niewygodne i pracochłonne, więc zainstalowano osobne perforator 5-kanałowy i 8-kanałowy, przełączane odpowiednimi włącznikami. Ponieważ przy tym urządzenia 8-kanałowe były uruchamiane oddzielnymi rozkazami, to można by uznać je za osobne urządzenia, jak gdyby wszystkie były dostępne jednocześnie. W emulatorze zostały zdefiniowane jako osobne i stale podłączone urządzenia, dzięki czemu, w odróżnieniu od rzeczywistej maszyny, można stosować w jednym programie naprzemiennie operacje we/wy 5- i 8-kanałowe. Schemat funkcjonalny komputera pokazuje kody operacji wejścia/wyjścia odnoszące się do poszczególnych urządzeń.

Do ODRY 1013 UMCS podłączono również głośnik, pełniący rolę monitora akustycznego. Zatrzymanie maszyny powodowało emisję ciągłego dźwięku z generatora akustycznego.

Głośnik można było przełączyć w ten sposób, że na zakończenie cykli pracy pętli głównej sterowania wydawał dźwięki o różnej wysokości, zależnie od szybkości pracy programu (od optymalności rozmieszczenia rozkazów) – najwyższe przy wykorzystaniu pamięci ferrytowej – co pozwalało na pisanie programów wygrywających proste melodie.

ODRA 1003 UMCS

Takiej maszyny nie było, ale ponieważ modyfikacja ODRY 1013 wykonana w Zakładzie Metod Numerycznych UMCS była niezależna od istnienia bądź nie pamięci ferrytowej, emulator oferuje również taką wersję. Mamy zatem maszyny:

ODRA 1003	– oryginalna, jak opisano wyżej
ODRA 1013	– oryginalna, jak opisano wyżej
ODRA 1013 UMCS	– zmodyfikowana, jak opisano wyżej
ODRA 1003 UMCS	– zmodyfikowana jak ODRA 1013 UMCS, ale bez pamięci ferrytowej

Monitor akustyczny

Wszystkie cztery wersje ODRY zostały w emulatorze wyposażone w opisany przy ODRZE 1013 UMCS głośnik, pełniący rolę monitora akustycznego, tj. służący do przywoływania operatora dźwiękiem po zatrzymaniu się maszyny.

Głośnik można włączać i wyłączać. Włączony emituje niedługi dźwięk *biiip* przy zatrzymaniu się maszyny z jakiegokolwiek powodu, ale tylko gdy ODRA pracuje w trybie ciągłym (tj. przy wciśniętym przycisku [PC] – Praca Ciągła). Przy pracy Start-Stopowej sygnał akustyczny nie rozlega się.

Emulator nie zawiera możliwości wygrywania na głośniku melodyjek w sposób zastosowany w ODRZE 1013 UMCS i opisany w [2].

O emulatorze

Emulator jest napisany w językach:

- HTML (Hyper Text Markup Language) – zgodnie ze standardem HTML5,
- CSS (Cascading Style Sheets) opisu wyglądu dokumentów HTML,
- JavaScript – oprogramowanie reagowania strony internetowej na zdarzenia

index.html – plik startowy Emulatora,
uruchamiany w folderze, w którym znajdują się pozostałe elementy oprogramowania

Ponieważ różnice między maszynami ODRA 1003, 1013, 1003 UMCS i 1013 UMCS są niewielkie, to program emuluje każdą z nich. Opis emulatora jest oparty na maszynie 1013 UMCS, ze wskazaniami ewentualnych różnic w stosunku do innych modeli.

Zgodnie z opisanym w [2] swobodnym sposobem podłączenia i sterowania urządzeń 8-kanałowych w maszynie UMCS, emulator ODRY UMCS ma pełny zestaw urządzeń we/wy, a nie zastępowanie jednych drugimi. Emulator wychwytuje rozkazy niezdefiniowane w danym modelu ODRY, ale zdefiniowane w innych, i przy wciśniętym przycisku [D] zatrzymuje maszynę.

Emulator nie emuluje pracy offline urządzeń we/wy. I tak:

- ✗ na dalekopisie nie można wykonywać operacji innych niż czytanie bądź pisanie w odpowiedzi na rozkazy komputera. Operacje offline (perforowanie tekstu na tasiemki, oddrukowywanie tekstów z tasiemek, reperforacja tasiemek z wprowadzaniem poprawek, itp.) bez udziału programu komputerowego nie są możliwe. Do ich wykonywania wystarczy zaskakująco prosty program czytający dane z dalekopisu (umieszczony w pamięci stałej). Odłączanie dalekopisu od komputera nie jest więc potrzebne i przycisk [D] (blokada dalekopisu) na pulpicie komputera stracił sens – został zaadaptowany (wraz z lampką KP) do wykrywania rozkazów niezdefiniowanych w danym modelu ODRY.
- ✗ Nie istnieje możliwość drukowania offline z tasiemki wykresów na samopisie MAW. Operację taką można wykonać tylko przy pomocy prostego (umieszczonego w pamięci stałej) programu przepisującego dane z czytnika taśmy 8-kanałowej na samopis MAW (sprzężony z nim perforator nie musi być wystartowany).

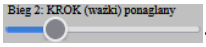
Na potrzeby emulacji przyjęto orientacyjnie następujące prędkości pracy urządzeń wejścia/wyjścia, nieco wyższe lub w górnym zakresie rzeczywistych:

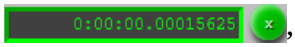
- czytnik taśmy perforowanej ~ 1066 rządków / s = (0,0009375 s = 6 kroków) / rządek
- perforator taśmy perforowanej 160 rządków / s = (0,00625 s = 40 kroków) / rządek
- samopis MAW 32 kropki / s = (0,03125 s = 200 kroków) / kropkę
- drukowanie na dalekopisie 16 znaków / s = (0,0625 s = 400 kroków) / znak
- pisanie na klawiaturze dalekopisu 8 znaków / s = (0,125 s = 800 kroków) / znak
- czytanie z tekstu w roli klawiatury 8 znaków / s = (0,125 s = 800 kroków) / znak

Emulator jest w pełni funkcjonalną maszyną, na której można wykonywać rzeczywiste programy, jak również śledzić na poziomie poszczególnych kroków procesora zawartość pamięci i rejestrów, oraz pracę procesora i urządzeń we/wy.

Panel emulatora

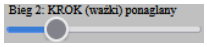
Na panelu emulatora znajdują się kontrolki sterujące procesem emulacji, oraz odwzorowanie pulpitu komputera ODRA 1003/1013. Kontrolki są, od lewej, następujące:

- Napis „**Eemc ODRA 1003/1013**” oznaczający „Emulator elektronicznej maszyny cyfrowej ODRA 1003 i 1013”.
- Przycisk otwierania i zamykania okienka podglądu procesora maszyny, pozwalającego analizować stan maszyny w najwolniejszych biegach emulatora.
- Przycisk otwierania i zamykania okienka podglądu pamięci maszyny, dającego szybką orientację w jej zawartości.
- Przycisk włączania i wyłączania głośnika będącego monitorem akustycznym. Włączony głośnik, po zatrzymaniu się z jakiegokolwiek powodu maszyny pracującej w sposób ciągły, przywołuje operatora dźwiękiem *biiip* do reakcji na zatrzymanie się programu.
- Przycisk ponaglenia służący do zezwalania na wykonanie następnego taktu, kroku lub cyklu (rozkazu) w biegach najwolniejszych (gdy następuje wstrzymywanie pracy emulatora). Przycisk ten jest normalnie nieaktywny; uaktywnia się tylko gdy emulator czeka na ponaglenie. Zmiana biegu na wystarczająco wyższy automatycznie ponagla emulator.
- Suwak prędkości emulacji 

.
- Zegar czasu 

, jaki rzeczywistej maszynie zajmowało wykonywanie programu. Czas pracy poszczególnych rozkazów jest naliczany z uwzględnieniem czekania na obrót bębna i szybkości urządzeń wejścia/wyjścia (czytniki, perforatory, samopis, dalekopis). Zegar ma przycisk zerowania wskazań.
- 

Logo modelu maszyny. Model maszyny można zmienić kliknięciem w emblemat, ale tylko przy wyłączonej maszynie. Ograniczenie wynika stąd, że pamięć ferrytowa jest adresowana jak gdyby była bębnową, ale bębnowa o tych samych adresach też istnieje i jest dostępna dla rozkazów przesyłania blokowego. Zmiana modelu przy włączonej maszynie mogłaby spowodować pobieranie danych z tej pamięci, w której ich nie ma.

Suwak prędkości emulacji 

zwalnia/przyspiesza pracę maszyny:

Bieg 0 – najwolniejszy, przewidziany do ewentualnego pokazywania pracy procesora na poziomie poszczególnych taktów. Aktualnie nie różni się od biegu 1, z jednym wyjątkiem: pozwala obserwować poszczególne sumowania w operacjach mnożenia jako półkroki (jedno sumowanie jako 20 taktów). W operacji mnożenia w jednym kroku wykonywały się jednocześnie dwa sumowania (przez dwa sumatory). Pokazanie każdego sumowania w oddzielnym półkroku pozwoliło zrezygnować z emulacji drugiego (nieopisanego) sumatora, a mimo to umożliwiło łatwą analizę algorytmu mnożenia.

Bieg 1 – w którym emulator wstrzymuje pracę po wykonaniu każdego kroku, umożliwiając analizę stanu komputera. Wykonanie następnego kroku wymaga ponaglenia przyciskiem  .

Bieg 2 – jest podobny do biegu 1, z tą różnicą, że następuje przeskok w czasie przez jałowe kroki oczekiwania na obrót bębna, tzn. wstrzymanie pracy następuje tylko po ważkich krokach. Czas kroków jałowych jest naliczany, ale poza tym wrażenie jest podobne jak przy pracy z pamięcią ferrytową.

Bieg 3 – emulator wstrzymuje pracę po wykonaniu każdego rozkazu (cyklu procesora). Wykonanie następnego rozkazu wymaga ponaglenia przyciskiem  .

Bieg 4 – ten i wyższe biegi powodują niewstrzymywaną pracę emulatora, a niebieski przycisk ponaglania pozostaje nieaktywny. Ta prędkość daje w zwolnionym tempie dobre wyobrażenie o pracy maszyny, pokazując wszystkie kroki, również jałowe.

Bieg 5 – równie powolny jak bieg 4, ale z przeskakiwaniem kroków jałowych i pokazywaniem jedynie kroków ważkich.

Bieg 6 – różni się od biegu 5 jedynie brakiem zwłoki między poszczególnymi, ważkimi krokami. Jest to najwyższa prędkość pokazująca stan maszyny co krok.

Bieg 7 – maksymalna prędkość emulatora pokazująca stan maszyny po każdym rozkazie (cyklu procesora).

Biegi 8 i 9 – prędkości „turbo” emulatora pokazujące stan procesora co któryś rozkaz, ale zachowujące w miarę naturalny widok pracującej maszyny. Biegi te mogą być przydatne do wykonywania wielogodzinnych obliczeń nawet kilka tysięcy razy szybciej od prawdziwej ODRY. Najwyższy bieg wymaga wystarczająco szybkiego komputera i wydajnego silnika JavaScript, na których uruchomiony jest emulator.

UWAGA: Operacja wykonywana przez urządzenie wejścia/wyjścia odbywa się bez wstrzymywania pracy emulatora po upływie każdego kroku, tj. tak, jak gdyby był tylko jeden długi krok, ale zegar odmierza [przyjęty] czas pracy rzeczywistego urządzenia.

Słowo o prędkości emulacji

Komputer ODRA 1003/1013 ma dwa tryby pracy: start-stopowy i ciągły. Normalnie, po naciśnięciu przycisku [StartCPU], [zA], [doA] lub [O], zapala się lampka StartCPU i zostaje wykonany rozkaz, po czym komputer zatrzymuje się, a lampka gaśnie. Wciśnięcie przycisku [PC] (praca ciągła) powoduje, że po wykonaniu rozkazu komputer nie zatrzymuje się, lecz przechodzi do wykonania następnego rozkazu, a zatrzyma się dopiero po naciśnięciu przycisku [StopCPU], powstaniu stanu powodującego zatrzymanie, wyłączeniu maszyny, lub tp. Podczas wykonywania rozkazu czytania maszyna może wejść w stan oczekiwania na urządzenie; wtedy rozkaz jest w trakcie wykonywania się i lampka się świeci.

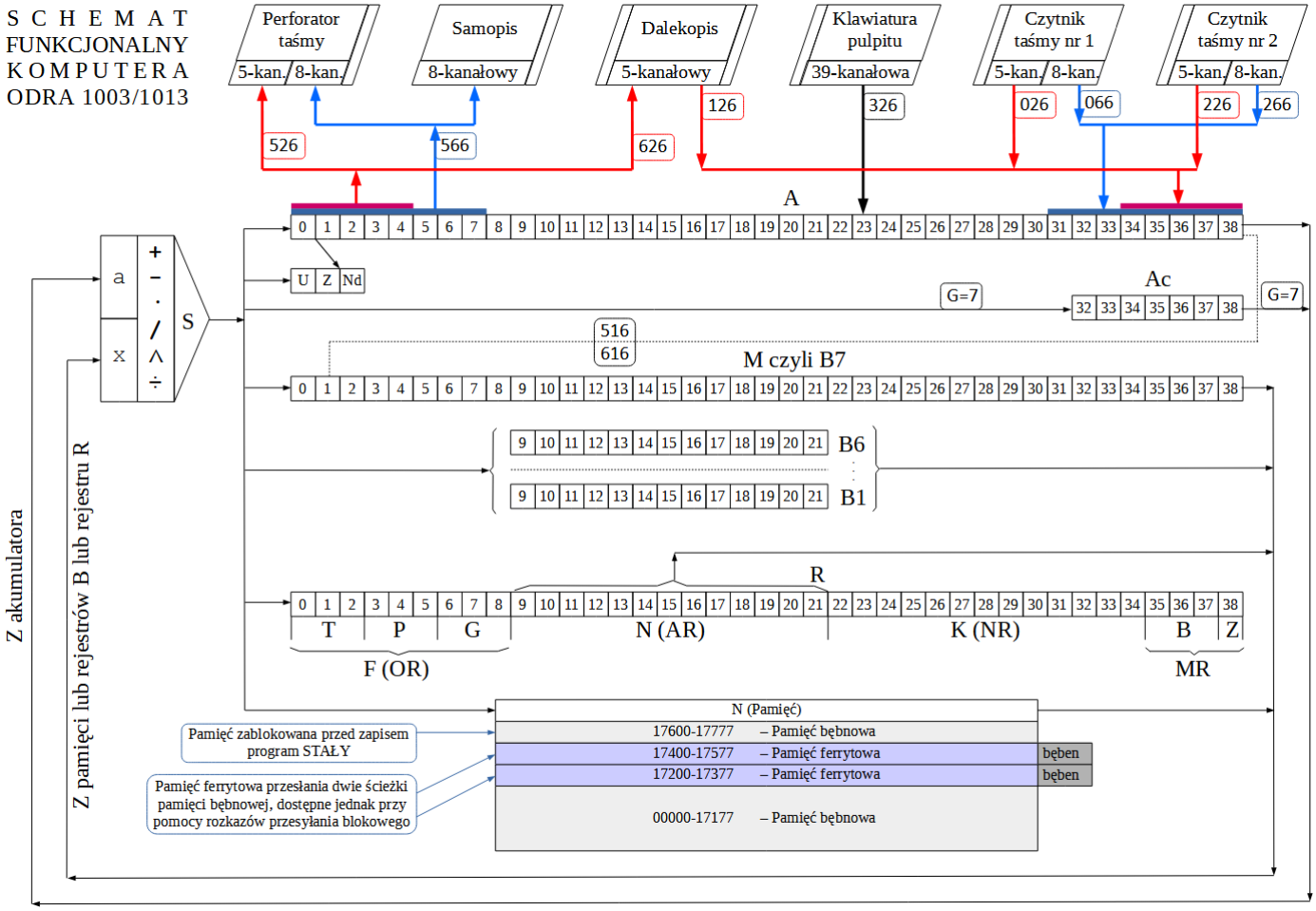
Z kolei suwak prędkości powoduje w najniższych biegach zatrzymywanie się emulatora do chwili ponaglenia, by po kolejnym takcie, kroku lub cyklu procesora znowu zatrzymać emulator.

Należy odróżnić zatrzymanie się emulatora od zatrzymania komputera. Emulator może zatrzymać się, chociaż komputer ciągle pracuje. Zatrzymanie emulatora najłatwiej rozumieć jako zatrzymanie czasu, przy czym przycisk ponaglenia  powoduje przeskok do następnej chwili, w której czas znowu jest zatrzymany. Jeżeli w tej nowej chwili rozkaz nie jest jeszcze wykonany do końca, to wymagane jest ponaglenie, by przeskoczyć do następnej chwili. Jeśli w danej chwili rozkaz jest wykonany do końca, to albo maszyna się zatrzymała (zgasła lampka Start) i do dalszej pracy konieczne jest naciśnięcie przycisku [StartCPU], albo maszyna pracuje w sposób ciągły (wciśnięty jest przycisk [PC] i świeci lampka Start) i do kontynuacji potrzebne jest tylko ponaglenie, by przeskoczyć w czasie do następnej chwili.

Regulowanie prędkości emulatora jest jak rozciąganie i ściskanie czasu

Jeśli więc, na małych prędkościach, widzimy że komputer niby pracuje, ale nic się nie dzieje, to prawdopodobnie z uwagi na zatrzymanie czasu suwakiem prędkości. Np. wciśnięcie przycisku [StopB], wyłączającego zasilanie, powoduje czekanie na wyłączenie się maszyny, ta na zatrzymanie się CPU, a CPU na zakończenie wykonywania się rozkazu, który wykona się, jeśli czas nie będzie zatrzymany.

Architektura ODRY 1013



Jednostka sterująca

Jednostka sterująca jest taktowana impulsami nagranymi na ścieżce zegarowej bębna magnetycznego, pełniącego rolę pamięci operacyjnej. Na jeden obrót bębna przypada 256000 taktów:

$$50 \text{ obrotów/s} \times 128 \text{ komórek/obrót} \times 40 \text{ taktów/komórkę} = 256000 \text{ Hz}$$

Jeden takt ścieżki zegarowej odpowiada przetworzeniu jednego bitu – jest to podstawowa częstotliwość taktowania procesora.

Czas obrotu bębna o jedną komórkę pamięci wynosi 0,00015625 s – jest to jednostka czasu wykonywania rozkazu, nazywana krokiem maszynowym. Wykonanie rozkazu pobieranego z pamięci (cykl rozkazowy) odbywa się zawsze w 4 etapach, na które składają się:

a) odszukanie rozkazu w pamięci (1 krok lub więcej),
b) pobranie rozkazu z pamięci z jednoczesnym wykonaniem B-modyfikacji (1 krok),
c) odszukanie lub przygotowanie argumentu rozkazu (1 krok lub więcej),
d) wykonanie rozkazu z ew. odczytem argumentu lub zapisem wyniku (1 krok lub więcej).

Rozkaz pochodzący z przycisku pulpitu maszyny [zA], [doA] lub [O], albo będący wynikiem obliczeń skierowanym do rejestru rozkazów, wykonuje się od etapu c). Co więcej, B-modyfikacja jest wykonywana jedynie podczas pobierania rozkazu z pamięci, w etapie b).

Przepływ informacji podczas wykonywania cyklu rozkazowego jest następujący:

A) w rejestrze aR znajduje się część NR ostatnio wykonanego rozkazu, albo adres ustawiony przyciskami pulpitu na części AR rejestru rozkazów R, gdy rozpoczynamy pracę od przycisku [StartCPU]. Jest to adres następnego rozkazu do wykonania. Następuje szukanie komórki pamięci o tym adresie. W rzeczywistej maszynie do wyszukiwania w pamięci służył rejestr aRw (wybierający), ale emulator nie pokazuje go. Adres zostaje podzielony na dwie części: 6 najstarszych bitów wyznacza numer ścieżki, a pozostałe 7 najmłodszych – numer strefy. Następuje oczekiwanie na pojawienie się pod głowicą czytającą wskazanej strefy bębna – są to jałowe kroki, w których odbywa się jedynie porównywanie wskazanego numeru strefy z numerami stref odczytywanych ze ścieżki adresowej, opisane jako „Czekanie na rozkaz”. Po odnalezieniu strefy zawartość rejestru aR przesyłana jest do rejestru SA i etap ten się kończy.

B) Ponieważ numer strefy jest zapisany z wyprzedzeniem o jedną strefę, to pod głowicę czytającą wchodzi właśnie komórka pamięci zawierająca rozkaz. Następuje „Pobieranie rozkazu”. Końcowe bity słowa są wczytywane jako pierwsze, dzięki czemu wykonuje się jednocześnie ew. B-modyfikacja polegająca na dodaniu do rozkazu zawartości rejestru B wskazanego w rozkazie. Dodawanie to jest wykonywane jak każde zwykłe dodawanie i odbywa się jedynie w trakcie pobierania rozkazu z pamięci operacyjnej. Treść rejestru B7, jako długa, pozwala zmodyfikować cały rozkaz, zawartość rejestru B1, B2, B3, B4 lub B5 dodaje się do części AR rozkazu (z ew. przeniesieniem na starsze bity), a zawartość rejestru B6 dodaje się do części NR rozkazu (też z ew. przeniesieniem). Formalnie można też używać nieistniejącego rejestru B0, stale zawierającego 0. Pobrany, zmodyfikowany (czyli skuteczny) rozkaz zostaje posłany do rejestru rozkazów R, a tam rozparcelowany na części. Część AR wczytanego rozkazu dostaje się do rejestru aR (i aRw) i etap ten kończy się po jednym kroku.

C) Jeśli wykonanie rozkazu będzie wymagało dostępu do pamięci, następuje szukanie komórki pamięci o adresie zawartym w rejestrze aR (informacja: „Czekanie na pamięć” – kroki jałowe, „Argument osiągnięty” – gdy będzie zapis do pamięci, „Argument pobrany” – gdy odczyt z pamięci), po czym argument wejściowy rozkazów arytmetycznych pojawia się na wejściu x sumatora. W rzeczywistej maszynie argument jest czytany dopiero w etapie d) podczas wykonywania operacji arytmetycznej, ale emulator ukazuje go na koniec tego etapu, gdyż potem mógłby pozostać niezauważony.

D) Czas wykonywania rozkazu zależy od jego treści. Na zakończenie cyklu rozkazowego część NR rozkazu jest posyłana do rejestru aR, a więc jednostka sterująca jest gotowa do wykonania następnego cyklu od etapu a). Szczególnym przypadkiem jest wykonanie rozkazu skoku pod adres zapisany na części AR rozkazu, który już znajduje się w rejestrze aR w wyniku wykonania etapu b) – wtedy części NR rozkazu nie posyła się do aR pozostawiając w nim potrzebny adres.

Zatrzymanie się maszyny może nastąpić tylko po zakończeniu wykonywania się rozkazu

Rejestry

Jednostka centralna ma rejestry 39-bitowe A, B7, oraz R. Każdy z nich ma również bit techniczny, normalnie niedostępny dla programu. Numeracja bitów rejestrów wynika stąd, że arytmometr działa tak, jak gdyby rejestr zawierał liczbę ułamkową, z przecinkiem tuż za bitem znaku.

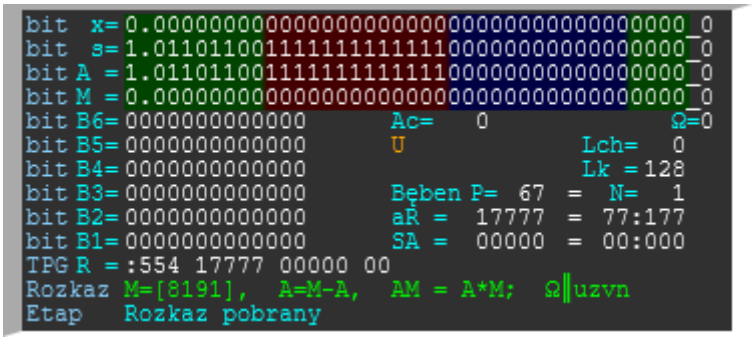
Jest również 6 rejestrów krótkich B1, B2, B3, B4, B5 i B6. Rejestry te w operacjach arytmetycznych zachowują się tak, jak gdyby miały 9 zer po lewej i 17 zer po prawej stronie. Służą one m. in. do tzw. B-modyfikacji rozkazów.

W operacjach na liczbach zmiennoprzecinkowych, cechę wyniku przechowuje się w rejestrze Ac. Liczba zmiennoprzecinkowa w pamięci lub rejestrze B ma cechę na bitach 32...38 słowa maszynowego, stąd taka a nie inna numeracja bitów rejestru Ac.

CPU ma też jednobitowe, normalnie niedostępne dla programisty, ale istotne, rejestry warunków: U, Z i Nd, oraz rejestr zaokrągleń Ω.

Inne rejestry maszyny pełnią wewnętrzną rolę, nieistotną na poziomie programu użytkowego.

Przycisk emulatora otwiera lub zamyka okienko, w którym można na bieżąco, krok po kroku, obserwować cykle pracy jednostki sterującej, pozycję głowic bębna, wartości podstawowych rejestrów maszyny, objaśnienie etapów cykli pracy i znaczenie wykonywanych rozkazów:



`x=` wartość na wejściu x głównego sumatora (drugi argument operacji). Na pierwsze wejście sumatora zawsze podawana jest zawartość akumulatora A. Komputer miał też drugi sumator, angażowany np. przy mnożeniu, ale z uwagi na jego bardzo elementarne zastosowanie, nie opisywane lecz tylko wzmiankowane, nie jest tu pokazany.

`s=` wartość wynikowa z sumatora. Wynik ten może być ponadto posłany do innego rejestru lub do pamięci operacyjnej.

`A =` zawartość rejestru akumulatora. W niektórych operacjach akumulator działa w parze z rejestrem M, jako swoim przedłużeniem – parę tę oznacza się AM. W operacjach zmiennoprzecinkowych akumulator zawiera mantysę liczby i wtedy jest nazywany rejestrem Am – parę rejestrów Am i Ac oznaczam symbolem AC.

`M =` zawartość rejestru B7 zwanego też mnożnikiem. W niektórych operacjach rejestr M działa jako przedłużenie akumulatora, przy czym bit znaku rejestru M nie wlicza się do przedłużenia i jest zerowany. Podczas B-modyfikacji rejestr B7 modyfikuje cały rozkaz.

`B6=` zawartość krótkiego rejestru B6. Podczas B-modyfikacji rejestr B6 modyfikuje drugą część adresową rozkazu.

`B5= B4= B3= B2= B1=` zawartości pozostałych krótkich rejestrów. Podczas B-modyfikacji rejestry te modyfikują pierwszą część adresową rozkazu.

`R =` zawartość rejestru rozkazów, czyli wykonywany rozkaz. W niektórych operacjach (mnożenia i dzielenia) jest to liczba przechowywana w R traktowanym jako rejestr roboczy.

`Rozkaz` zapis funkcji wykonywanego rozkazu maszynowego w postaci instrukcji podobnych do języka C lub JavaScript. Po średniku kończącym instrukcje znajduje się symboliczne wskazanie ew. ustawiania bitu zaokrągleń Ω i warunków.

`Etap` wskazanie etapu cyklu pracy procesora, objaśnienie wykonywanej czynności.

`Ac=` zawartość 7-bitowego rejestru Ac cechy liczby zmiennoprzecinkowej.

`Ω=` zawartość 1-bitowego rejestru zaokrągleń.

`U Z Nd` zespół warunków: Ujemne, Zerowe, Nadmiar stałoprzecinkowy, ustawianych i testowanych przez rozkazy maszynowe.

`Lch=` licznik chwil (taktów) procesora, wykonanych w ramach kroku procesora.

`NZ KP` zespół warunków: Nadmiar Zmiennoprzecinkowy, Kontrola Komputer-Program, powodujących zatrzymanie się maszyny.

`Lk=` licznik kroków wykonanych w ramach cyklu rozkazowego procesora.

`Bęben P=ppp = N=nnn` numer strefy bębna magnetycznego, znajdującej się przed głowicą czytająco-piszącą, tj. w pozycji gotowej do wykonania odczytu lub zapisu w pamięci operacyjnej. Wartość `P` jest numerem wg adresacji z przeplotem co 7 stref, wartość `N` to numer wg adresacji normalnej co 1. Oba numery podane są ósemkowo.

`aR = aaaaa = gg:sss` zawartość rejestru wewnętrznego procesora. Rejestr ten najpierw zawiera adres rozkazu do pobrania, a potem adres argumentu wejściowego lub wyjściowego. Wyświetlony tu adres `aaaaa` jest podany również z podziałem na 6-bitowy numer ścieżki (głowicy) pamięci bębnowej, oraz 7-bitowy numer strefy bębna (wszystko ósemkowo).

`SA = aaaaa = gg:sss` zawartość (ósemkowo) rejestru SA, w którym znajduje się adres pobieranego i wykonywanego rozkazu. Adres ten jest też wyświetlany na lampkach pulpitu maszyny.

Okno podglądu nie ukazuje innych rejestrów pomocniczych jak oR, nR, mR, zR i aRw, układu czasu T, rejestru P, rejestru pośredniczącego cechy Pc, ani zawsze wyzerowanego (nieistniejącego) rejestru B0.

Rejestry długie: x, s, A, M i R, mają bit techniczny, najmłodszy, ukazany w formacie bitowym po znaku podkreślenia. Bit ten jest niemal zawsze zerowy – uczestniczy jedynie w operacjach mnożenia i dzielenia (stało- i zmiennoprzecinkowego).

Zawartość rejestru może być wyświetlana w różnych formatach. Zmiany formatu dokonuje się przez kliknięcie w napis formatu. Po najechaniu kursorem na nazwę rejestru, jego zawartość ukazuje się w dymku we wszystkich dostępnych formatach (tylko w trzech pierwszych dla rejestrów krótkich):

`bit` jako ciąg bitów; w przypadku rejestrów długich z kropką po bicie znaku i bitem technicznym po znaku podkreślenia, oraz kolorowym tłem ułatwiającym czytanie grup bitów,

`oct` jako ciąg cyfr ósemkowych,

`nat` jako liczba naturalna, dziesiętnie,

`int` jako liczba całkowita, dziesiętnie,

`dec` jako liczba ułamkowa z zakresu [-1, 1), dziesiętnie,

`flo` jako liczba zmiennoprzecinkowa, z mantysą na bitach 0..31, a cechą na bitach 32.. 38 słowa. W przypadku A mantysę stanowi cała zawartość rejestru, a cechę zawartość rejestru Ac.

`txt` jako ciągi siedmiu znaków ITA2 na bitach 0..34 (kolejne ciągi w kolejnych możliwych pocztach kodu dalekopisowego),

`TPG` jako rozkaz maszynowy w postaci :TPG N K BZ przyjętej w języku PODSTAWOWYM,

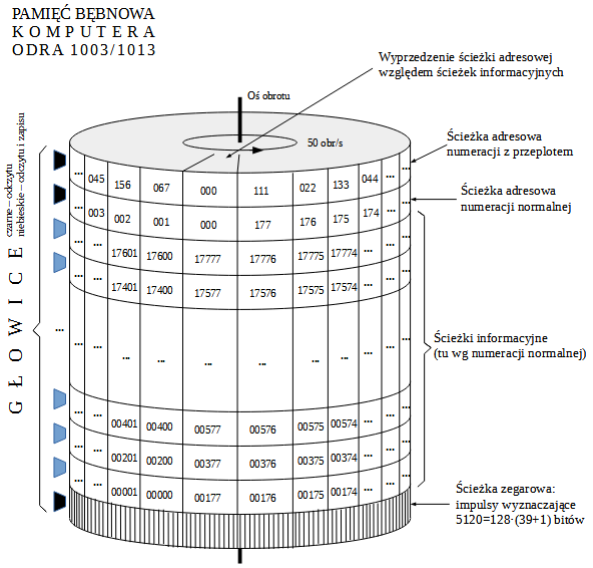
`Roz` jako objaśnienie rozkazu maszynowego, przy pomocy instrukcji C-podobnych.

UWAGA 1: W rejestrze R widać skuteczną wartość rozkazu, tj. po ew. B-modyfikacji wykonywanej w trakcie pobierania rozkazu z pamięci; w pozostałych rejestrach lub w pamięci jest to wartość przed B-modyfikacją.

UWAGA 2: B-modyfikacja rejestrem B7 modyfikuje cały rozkaz, w tym jego część MR, zatem przed dodaniem wartości B7 nie wiadomo, jaka będzie ostatecznie część MR, czyli który rejestr B będzie ostatecznie uczestniczył w operacji. Emulator zwraca na to uwagę wskazując wtedy w objaśnieniu rozkazu oznaczenie B? w miejsce B7.

Pamięć operacyjna

Pamięć operacyjną ODRY 1003 stanowi bęben magnetyczny o pojemności 64 ścieżek po 128 słów maszynowych. Ponadto są tam dwie ścieżki adresowe: dla numeracji komórek normalnej (co 1) oraz z przeplotem (co 7), a także ścieżka zegarowa taktująca procesor. Jedna komórka pamięci zawiera 40 bitów (39 bitów danych, ponumerowanych od 0 do 38, oraz jeden bit techniczny). Jeden takt ścieżki zegarowej odpowiada jednemu bitowi. Bęben obraca się z częstotliwością 50 obrotów/s, zatem podstawowa częstotliwość taktowania procesora wynosi:



50 obrotów/s × 128 komórek/obrót × 40 taktów/komórkę = 256000 Hz

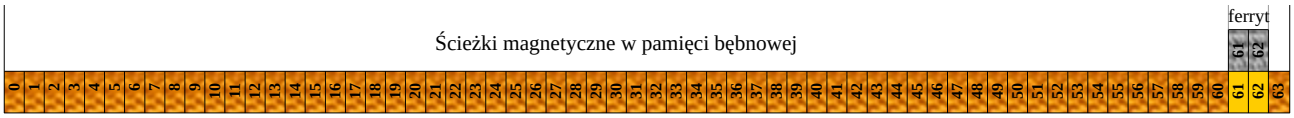
Numer strefy na ścieżce adresowej zapisany jest z wyprzedzeniem o jedną strefę w stosunku do adresowanej komórki pamięci.

Odczyt lub zapis w pamięci wymaga:

- odczekania, aż numer właściwej strefy pojawi się pod głowicą czytająco-piszącą,
- odczytania tego numeru, by stwierdzić, że jest to ten o który chodzi – w trakcie tego odczytu bęben obróci się do następnej strefy, w której znajduje się szukana komórka pamięci,
- odczytania lub zapisania treści komórki pamięci, w trakcie czego bęben obróci się znowu o jedną strefę.

Stąd wynika, że przy optymalnym rozłożeniu w pamięci rozkazów i argumentów, najszybszy rozkaz wykonuje się w 4 krokach, co daje maksymalną prędkość maszyny 1600 rozkazów/s.

ODRA 1013 jako pamięć operacyjną ma, oprócz bębnowej, pamięć ferrytową o pojemności 256 słów, oraz 4 dodatkowe rozkazy szybkiego przesyłania blokowego zawartości ścieżek bębnowych na ferrytowe i odwrotnie. Pamięć ferrytowa pracuje jak dwie (2 × 128 słów) ścieżki bębnowe:
ścieżka nr $61_{(10)}=75_{(8)}$ – zawierająca komórki o adresach $17200_{(8)} – 17377_{(8)}$ $(7808_{(10)} – 7935_{(10)})$,
i ścieżka nr $62_{(10)}=76_{(8)}$ – zawierająca komórki o adresach $17400_{(8)} – 17577_{(8)}$ $(7936_{(10)} – 8063_{(10)})$.



Ścieżki pamięci ferrytowej nie zastępują ścieżek bębnowych o adresach $17200_{(8)}$ do $17577_{(8)}$, a tylko je przesłaniają. Odwołanie się w rozkazie do komórki z tego zakresu jest odwołaniem się do pamięci ferrytowej, a spoza tego zakresu – do pamięci bębnowej. Wyjątkiem są rozkazy przesyłania blokowego pomiędzy tymi pamięciami. Przesłanie danych z pamięci ferrytowej pod adres bębnowy z zakresu ferrytowego, przesyła na odpowiednią ścieżkę bębnową, normalnie zasłoniętą dla innych rozkazów. Podobnie odwrotnie. Da się przesyłać blokowo dane pomiędzy ścieżkami ferrytowymi, a zasłoniętymi przez nie bębnowymi. Zasłonięte ferrytem ścieżki bębnowe mogą więc posłużyć jako schowek.

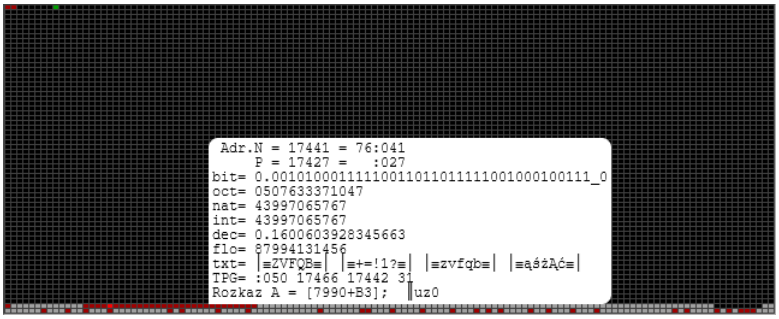
Odszukanie komórki w pamięci ferrytowej jest praktycznie natychmiastowe, stąd etapy a) i c) trwają zawsze 1 krok. Dzięki temu uzyskuje się w praktyce dwukrotnie większą szybkość pracy od ODRY 1003.

Ścieżki ferrytowe tracą zawartość z chwilą wyłączenia maszyny – bębnowe zachowują zapis po wyłączeniu maszyny i zasilania, aż do odświeżenia lub zamknięcia strony emulatora.

Ostatnia ścieżka pamięci bębnowej, adresy od $17600_{(8)}$ do $17777_{(8)}$, zawiera program STAŁY, nazwany tak, gdyż zapis na tę ścieżkę jest zablokowany sprzętowo. Emulator pozwala jednak pisać na nią po wybraniu modelu maszyny ODRA 1003 (następnie można zmienić model komputera).

Przycisk  emulatora otwiera lub zamyka okienko podglądu pamięci. Dymek ostrym rogiem wskazuje komórkę, której treść wyświetla szczegółowo. Kolor komórki sugeruje typ jej wartości – podczas pobierania lub zapisywania treści następuje rozjaśnienie koloru:

– komórka nie była używana,
– coś zostało zapisane,
– liczba zmiennoprzecinkowa,
– rozkaz maszynowy,
– ślad z podprogramu.



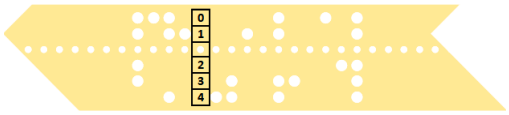
Kliknięcie myszki w komórkę cumuje do niej dymek, a kliknięcie w dymek odcumowuje. Nieprzycumowany dymek znika po przesunięciu kursora myszki poza obszar pamięci. Położenie widocznego, przycumowanego dymku można precyzyjnie korygować klawiszami strzałek.

Urządzenia We/Wy

Emulator nie obsługuje żadnych rzeczywistych urządzeń, z wyjątkiem możliwości pisania na dalekopisie za pomocą klawiatury komputera PC. Za nośniki danych służą odpowiednie pliki komputera-gospodarza.

Kod ITA2

Podstawowym urządzeniem wejścia/wyjścia znakowego, służącego do komunikacji z człowiekiem, jest dalekopis zgodny ze standardem kodowania ITA2 (Międzynarodowym Kodem Dalekopisowym nr 2) – 5-bitowym kodem stosowanym w telekomunikacji. Jako nośnik informacji służyły papierowe taśmy perforowane:



Pojedynczy poprzeczny rządek perforacji (kwintet) oznacza jeden znak kodu, przy czym jego interpretacja znakowa zależy od wcześniej umieszczonych znaków zmiany pocztu. Np. drugi od lewej kwintet (o wartości binarnej `000·01`) oznacza albo literę E, albo cyfrę 3. W kodzie ITA2 są dwa poczty znakowe: poczet wielkich liter (W) i poczet cyfr (C). Kod ten modyfikowano przez podkładanie innych znaków w miejsce zdefiniowanych, dodanie trzeciego pocztu – pocztu małych liter (m) lub np. cyrylicy, czy nawet czwartego – pocztu znaków specjalnych (s).

MIĘDZYNARODOWY KOD DALEKOPISOWY NR 2

Kanał 1: 1 Kanał 2: 2 Kanał 3: 4 Kanał 4: 8 Kanał 5: 16		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Glify CCITT		NU □		LF ≡		SP △	@ 品			CR ⇐	* ✠		; 品																FS ⇧				LS ⇩
GB	W:	●	E	≡	A		S	I	U	⌂	D	R	J	N	F	C	K	T	Z	L	W	H	Y	P	Q	O	B	G	▲	M	X	V	▼
	C:	●	3	≡	-		'	8	7	⌂	*	4	;	,	!	%	:	(| 5 | + |)	2	£	6	0	1	9	?	&	@	▲	.	/	=	▼
	m:	●	e	≡	a		s	i	u	⌂	d	r	j	n	f	c	k	t	z	l	w	h	y	p	q	o	b	g	▲	m	x	v	▼
	s:	●	Ş	≡	_		#	»	«	⌂	`	$	µ	<	\	^	[	{	±	]	€	@	}	÷	~	°	¢	%	▲	>	|	"	▼
US	C:	●	3	≡	-		@	8	7	⌂	$	4	'	,	!	:	(	5	"	)	2	#	6	0	1	9	?	&	▲	.	/	;	▼
FR	C:	●	3	≡	-		'	8	7	⌂	*	4	;	,	É	:	(	5	+	)	2	£	6	0	1	9	?	%	▲	.	/	=	▼
DE	C:	●	3	≡	-		'	8	7	⌂	*	4	;	,	Ä	:	(	5	+	)	2	Ü	6	0	1	9	?	Ö	▲	.	/	=	▼
SC	C:	●	3	≡	-		'	8	7	⌂	*	4	;	,	Å	:	(	5	+	)	2	Ö	6	0	1	9	?	Ä	▲	.	/	=	▼
RU	C:	●	3	≡	-		'	8	7	⌂	*	Ч	Ю	,	Э	:	(	5	+	)	2	Щ	6	0	1	9	?	Ш	▲	.	/	=	▼
	m:	●	Е	≡	А		С	И	У	⌂	Д	Р	Й	Н	Ф	Ц	К	Т	З	Л	В	Х	Ы	П	Я	О	Б	Г	▲	М	Ь	Ж	▼
PL	C:	●	3	≡	-		'	8	7	⌂	*	4	;	,	!	:	(	5	+	)	2	Ł	6	0	1	9	?	&	▲	.	/	=	▼
	s:	●	ę	≡	_		#	Ź	Ś	⌂	ź	Ę	ń	<	ż	^	[	Ń	ą	]	Ć	ł	Ó	Ż	Ą	ó	ć	%	▲	>	|	ś	▼

UWAGA: Niektóre znaki ze względów praktycznych, np. nieprzewidywalną szerokość glifów, mają w emulatorze inną reprezentację graficzną (zielone tło) niż w zaleceniach CCITT. Spację SP wskazuje się też znakiem ␣.

Beżowym tłem zaznaczono narodowe modyfikacje kodu. W standardzie brytyjskim, zamiast znaków ! i & używano też % i @. W innych językach podkładano znaki narodowe w miejsce znaków rzadko używanych. W standardzie rosyjskim cyfra 4 pełniła rolę litery Ч.

Czwarte poczty znaków (s), podane tu przykładowo dla standardu brytyjskiego i polskiego (żółte tło), są nieznane i zostały opracowane przeze mnie.

Znaki SP (◡ SPace – odstęp), CR (⌂ Carriage Return – powrót karetki na początek wiersza), oraz LF (≡ Line Feed – wysuw papieru o jeden wiersz) mają jednakowe znaczenie w każdym poczcie. Znaki NU (● Null – pusty) znajdowały się na początku i końcu taśmy perforowanej. Znaki FS (▲ Figure Shift – przełączenie na cyfry) i LS (▼ Letter Shift – przełączenie na litery) zostały przeznaczone w kodzie ITA2 do zmiany pocztów znakowych.

Interpretacja kodów na taśmie perforowanej zawierającej tekst powinna być zgodna ze sposobem działania dalekopisu podczas przepisywania treści z taśmy na wydruk (stąd odwołania do dalekopisu również przy dyskusji o taśmach).

Konwencje przełączania pocztów i tryby pracy urządzenia (dalekopisu)

Dla dalekopisów oferujących więcej niż dwa poczty znaków, zwykle dodatkowo trzeci (poczet małych liter), wypracowano konwencje przełączania większej liczby pocztów. Niektóre z nich nie były zgodne (w dół) z działaniem dalekopisów z dwoma pocztami. W emulatorze oprogramowano konwencje najbardziej znane i kompatybilne w dół.

Dalekopis z dwoma pocztami znaków

Początkowo dalekopis pracuje w trybie wielkich liter, interpretując kwintety jako litery z pocztu wielkich liter (W). Znaki NU (●) nie zmieniają pocztu.

Znaki FS (▲) i LS (▼) przełączają dalekopis odpowiednio na poczet cyfr i liter. Każdy z tych znaków mógł być użyty wielokrotnie bez wpływu na wydruk, i tak np. zbędne znaki LS służyły do zamaskowania błędów na taśmie perforowanej (zadziurkowania).

Dalekopisy z wieloma pocztami potrafią działać tak, jak urządzenie z dwoma pocztami, z tym ograniczeniem, że zbędny kod LS lub FS może spowodować przełączenie pocztu i zmianę trybu pracy na taki, w którym kody LS i FS uzyskują nowe znaczenie, na:

Tryb Małych Liter – sposób pracy urządzenia z większą liczbą pocztów znakowych,
Tryb Wielkich Liter – sposób pracy urządzenia z jedynie dwoma pocztami znaków.

Dalekopis z trzema pocztami znaków

Bezpośrednio po włączeniu zasilania dalekopis działa w trybie wielkich liter, interpretując kwintety jako litery z pocztu wielkich liter (W). Zbędny kod FS (▲) nie ma specjalnego znaczenia, natomiast zbędny kod LS (▼) przełącza dalekopis na poczet małych liter (m) w trybie małych liter.

W trybie małych liter kod FS (▲) przełącza na poczet cyfr, a po napisaniu znaków cyfrowych, kod LS (▼) powraca do pocztu małych liter, pozostawiając urządzenie w trybie małych liter. W poczcie małych liter kod LS (▼) przełącza na poczet wielkich liter tylko na czas jednego następnego kodu, po czym dalekopis automatycznie powraca do pocztu małych liter, pozostając w trybie małych liter (nie oprogramowano tu interpretacji, że przełącza na stałe na poczet wielkich liter).

Dalekopis z czterema pocztami znaków

Działa jak dalekopis z trzema pocztami, ale teraz zbędny kod FS (▲) przełącza na poczet (s) znaków specjalnych na czas jednego znaku, i automatycznie powraca do pocztu cyfr, pozostawiając tryb małych liter (nie oprogramowano tu interpretacji, że przełącza na stałe na czwarty poczet).

Wyjście z trybu małych liter do trybu wielkich liter umożliwia jedynie kombinacja FS+LS (▲▼), w której kod LS następuje bezpośrednio po kodzie FS.

W kodzie rosyjskim w poczcie małych liter znajdują się znaki cyrylicy i prócz LS (▼) i FS (▲) przełączających jak w trybie wielkich liter, kod NU (●) dodatkowo przełącza dalekopis na alfabet rosyjski, dzięki czemu nie ma potrzeby rozróżniania trybów pracy urządzenia.

Dalekopis działający w trybie wielkich liter może mieć dodatkowe poczty, ale ich nie używa dopóki nie zostanie przełączony w tryb małych liter. Oznacza to, że nie należy beztrosko stosować zbędnych znaków FS i LS, by przypadkowo nie wytrącić urządzenia z właściwego trybu pracy (na szczęście grozi to zwykle tylko zamianą małych liter na wielkie lub odwrotnie, albo dotyczy tylko jednego znaku z czwartego pocztu).

Sterowanie dalekopisem z dwoma pocztami znaków

Do przełączania trybów pracy i pocztów znakowych emulator stosuje tablice krosowe stanów dalekopisu. Na początku, tj. bezpośrednio po włączeniu zasilania, dalekopis pozostaje w stanie 0. W każdym stanie s (wierszu numer s tablicy krosowej) zdefiniowane są pary (p1, s1) przejść do nowego stanu s1, w których p1 wyznacza poczet wg którego będzie interpretowany następny kod ITA. Wejściowy kod ITA decyduje, która para (p1, s1) bieżącego stanu zostanie wybrana jako przejście do nowego stanu. Wybierana jest para z kolumny odpowiadającej kodowi znaku LS (▼), FS (▲), NU (●), lub zwykłego znaku (czyli innego, w tym CR ⌂, LF ≡ i SP ◡). Liczba stanów zależy od typu dalekopisu, natomiast poczty oznaczmy literami:

`ABC` – poczet wielkich liter (W) albo: `Abc` – by podkreślić tryb małych liter
`123` – poczet cyfr (znaków figurowych) (C) albo: `₁₂₃` – by podkreślić tryb małych liter
`abc` – poczet małych liter (m) albo: `ДИЯ` – poczet liter cyrylicy (m)
`Ż₂₃` – poczet znaków specjalnych (s) z podkreśleniem trybu małych liter

ABC123	Stary stan dalekopisu	Nowy (poczet, stan) po kodzie wejściowym:			
		Zwykły znak	LS ▼	FS ▲	NU ●
Litery (stan początkowy)	0	(ABC, 0)	(ABC, 0)	(123, 1)	(ABC, 0)
Cyfry	1	(123, 1)	(ABC, 0)	(123, 1)	(123, 1)

Sterowanie dalekopisem z trzema pocztami znaków

Stosują się tu objaśnienia dane przy opisie dalekopisu z dwoma pocztami.

ABC123abc	Stary stan dalekopisu	Nowy (poczet, stan) po kodzie wejściowym:			
		Zwykły znak	LS ▼	FS ▲	NU ●
Litery Wielkie (stan początkowy)	0	(ABC, 0)	(abc, 2)	(123, 1)	(ABC, 0)
Cyfry	1	(123, 1)	(ABC, 0)	(123, 1)	(123, 1)
Litery małe	2	(abc, 2)	(Abc, 2)	(₁₂₃, 4)	(abc, 2)
Cyfry w trybie małych liter	3	(₁₂₃, 3)	(abc, 2)	(₁₂₃, 4)	(₁₂₃, 3)
Stan rozterki po „abc▲”	4	(₁₂₃, 3)	(ABC, 0)	(₁₂₃, 4)	(₁₂₃, 3)

Ze względu na trudniejsze sterowanie, w praktyce nie zwracano większej uwagi na przełączanie liter wielkich i małych, a czwartego pocztu prawdopodobnie wcale nie było.

Sterowanie dalekopisem z czterema pocztami znaków

Stosują się tu objaśnienia dane przy opisie dalekopisu z dwoma pocztami.

ABC123abcŻĄĆ	Stary stan dalekopisu	Nowy (poczet, stan) po kodzie wejściowym:			
		Zwykły znak	LS ▼	FS ▲	NU ●
Litery Wielkie (stan początkowy)	0	(ABC, 0)	(abc, 2)	(123, 1)	(ABC, 0)
Cyfry	1	(123, 1)	(ABC, 0)	(Ż$_{23}$, 5)	(123, 1)
Litery małe	2	(abc, 2)	(Abc, 2)	($_{123}$, 4)	(abc, 2)
Cyfry w trybie małych liter	3	($_{123}$, 3)	(abc, 2)	(Ż$_{23}$, 5)	($_{123}$, 3)
Stan rozterki po „abc▲”	4	($_{123}$, 3)	(ABC, 0)	(Ż$_{23}$, 4)	($_{123}$, 3)
Stan rozterki po „123▲”	5	($_{123}$, 3)	(ABC, 0)	(Ż$_{23}$, 5)	($_{123}$, 3)

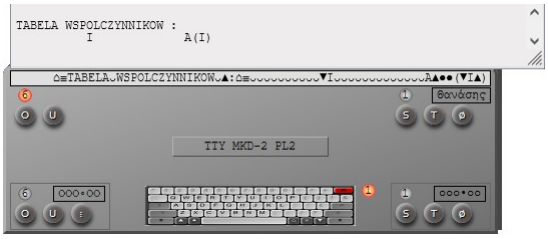
Stany 5 i 6 są tu podobne do siebie, ale ich rozróżnienie jest potrzebne dla ew. automatycznego generowania kodów przełączających LS (▼), FS (▲) i NU (●), na podstawie znaków Unicode.

Sterowanie dalekopisem z trzema pocztami wg konwencji rosyjskiej

Tu sterowanie jest bardzo proste (analogiczne do sterowania dalekopisu z dwoma pocztami) dzięki wykorzystaniu kodu NU (●), który użyty w dużej liczbie mógł sprawiać kłopoty techniczne w telekomunikacji, ale przy podłączeniu krótkim kablem do komputera nie powodował przekłamań.

ABC123ДИЯ	Stary stan dalekopisu	Nowy (poczet, stan) po kodzie wejściowym:			
		Zwykły znak	LS ▼	FS ▲	NU ●
Litery łacińskie (stan początkowy)	0	(ABC, 0)	(ABC, 0)	(123, 1)	(ДИЯ, 2)
Cyfry	1	(123, 1)	(ABC, 0)	(123, 1)	(ДИЯ, 2)
Litery cyrylicy	2	(ДИЯ, 2)	(ABC, 0)	(123, 1)	(ДИЯ, 2)

Dalekopis



Wydruki z dalekopisu są tworzone w postaci czystego pliku tekstowego w kodzie UTF-8. Znak LF powoduje przejście do początku nowego wiersza, zaś znak CR nie cofa karetki do początku tego samego wiersza, uniemożliwiając pisanie po wcześniejszym tekście.

W odróżnieniu od rzeczywistego urządzenia, dalekopis emulatora nie może pracować offline; jest urządzeniem odpowiadającym na wysyłane z komputera rozkazy czytania i pisania. Są w nim:

- tabliczka identyfikacyjna typu dalekopisu,
- klawiatura działająca w trybie Baudot lub w trybie PC (lampka z numerem adresowym i wyobrażenie klawiatury służące do przywoływania wirtualnej klawiatury ekranowej podwójnym kliknięciem),
- wejście z pliku tekstowego działające zamiennie z klawiaturą (lampka z numerem adresowym, prostokątne pole ukazujące ostatnio wczytane znaki i służące do podglądania pliku, przyciski analogiczne jak w czytniku taśmy perforowanej),

- 5-kanałowy czytnik taśmy perforowanej (podobny do innych czytników, ale niewidoczny dla komputera),
- perforator taśmy 5-kanałowej (podobny do innych, ale niewidoczny dla komputera),
- drukarka (lampka z numerem adresowym, szerokie pole wyobrażające wiersz drukarki i przywołujące podgląd wydruku, przyciski (O) i (U) analogiczne jak w perforatorze),
- monitor danych wchodzących do dalekopisu, umieszczony na polu przywołującym podgląd wydruku. Monitor został pomyślany jako edukacyjna wizualizacja danych, gdyż nie istniał w rzeczywistych urządzeniach.

Na wejściu do dalekopisu pojawiają się kwintety pochodzące w danej chwili albo z rozkazu pisania komputera, albo w odpowiedzi na rozkaz czytania z komputera:

- z klawiatury wirtualnej na ekranie lub rzeczywistej komputera-gospodarza,
- lub z pliku tekstowego „wyręczającego” klawiaturę,
- lub z czytnika taśmy perforowanej dalekopisu.

Przyciskami (S) można dynamicznie przełączać wejście dalekopisu: klawiatura / tekst / czytnik.

Każdy kwintet wejściowy ukazuje się w okienku monitora i zmienia stan dalekopisu zgodnie z jego typem. Bieżący stan dalekopisu decyduje o interpretacji kolejnego kodu wejściowego, tj. wybraniu pocztu znakowego i zmianie stanu. Wydruk powstaje zgodnie ze stanem dalekopisu.

Na wyjściu dalekopisu drukarka i perforator pracują niezależnie od siebie. Na te urządzenia kierowane są kwintety pojawiające się na wejściu dalekopisu, tj. pojawiające się w okienku monitora. Przyciskami (O) można włączać i wyłączać drukowanie i/lub perforowanie, zatem kody trafiające do dalekopisu mogą wychodzić na oba wyjścia, na jedno z dwojga, albo trafiać zupełnie w nicość. Wyłączanie i włączanie drukowania nie wpływa na stan dalekopisu, a więc nie zakłóca wyboru pocztu znakowego.

Urządzenia zachowują się zgodnie z rzeczywistymi – np. zamontowanie taśmy w czytniku z pominięciem kodów przełączających poczty znakowe może spowodować niewłaściwą interpretację kwintetów. Podobne efekty mogą wystąpić przy przełączaniu wejścia klawiatura / tekst / czytnik.

Typ dalekopisu

Emulator umożliwia wybranie jednego z kilku zdefiniowanych typów dalekopisu. Typ dalekopisu charakteryzuje jego nazwa wyświetlana na tabliczce identyfikacyjnej, repertuar znaków dostępnych w poszczególnych pocztach znakowych, oraz sposób przełączania pocztów. Podpowiedź, jak są przełączane poczty znaków, ukazuje się jako bryk w dymku pod tabliczką identyfikacyjną.

Zmiany typu dalekopisu można dokonać podwójnym kliknięciem w tabliczkę identyfikacyjną, ale wyłącznie przy wyłączonym zasilaniu (przycisk [StopB]). Ograniczenie wynika stąd, że kody ITA2 zmieniają stan urządzenia zgodnie z jego typem. Stany dalekopisu jednego typu nie muszą pokrywać się ze stanami innego typu, więc zmiana typu w trakcie pracy mogłaby zaburzyć działanie urządzenia. Po zmianie modelu dalekopisu stan dalekopisu jest resetowany, ale wszystkie urządzenia we/wy zachowują dane i pozycję.

W emulatorze zdefiniowano kilka typów dalekopisów, z różnymi alfabetami, wśród których znajdują się urządzenia z każdym opisanym wcześniej sterowaniem.

Wykaz zdefiniowanych w emulatorze dalekopisów

Nazwa	Sterowanie	Poczet	Alfabet
MKD-2 PL2	ABC123	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,!:(5+)2Ł6019?&▲./=▼
MKD-2 PL3	ABC123abc	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,!:(5+)2Ł6019?&▲./=▼
		abc	●e≡a◡siu⌂drjnfcktzlwhypqobg▲mxv▼
MKD-2 PL4	ABC123abcŻĄĆ	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,!:(5+)2Ł6019?&▲./=▼
		abc	●e≡a◡siu⌂drjnfcktzlwhypqobg▲mxv▼
		ŻĄĆ	●ę≡_◡#ŹŚ⌂źĘń<ż^[Ńą]ĆŁÓŻĄóć%▲>|ś▼
ITA2 GB2	ABC123	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,!:(5+)2£6019?&▲./=▼
ITA2 GB3	ABC123abc	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,!:(5+)2£6019?&▲./=▼
		abc	●e≡a◡siu⌂drjnfcktzlwhypqobg▲mxv▼
ITA2 GB4	ABC123abcŻĄĆ	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,!:(5+)2£6019?&▲./=▼
		abc	●e≡a◡siu⌂drjnfcktzlwhypqobg▲mxv▼
		ŻĄĆ	●§≡_◡#»«⌂`$µ<\^[{±]€@}÷~°¢%▲>|"▼
ITA2 DE2	ABC123	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,Ä:(5+)2Ü6019?Ö▲./=▼
ITA2 FR2	ABC123	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,É:(5+)2£6019?%▲./=▼
ITA2 SC2	ABC123	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*4;,Å:(5+)2Ö6019?Ä▲./=▼
ITA2 US2	ABC123	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡@87⌂$4',!:(5")2#6019?&▲./;▼
МТК-2 РУ3	ABC123ДИЯ	ABC	●E≡A◡SIU⌂DRJNFCKTZLWHYPQOBG▲MXV▼
		123	●3≡-◡'87⌂*ЧЮ,Э:(5+)2Щ6019?Ш▲./=▼
		ДИЯ	●E≡A◡СИУ⌂ДРЙНФЦКТЗЛВХЫПЯОБГ▲МЬЖ▼

Klawiatura dalekopisu

Oryginalna klawiatura dalekopisu miała 26 klawiszy z literami i cyframi, oraz 6 klawiszy: SP, CR, LF, NU, FS i LS. Każde naciśnięcie klawisza wprowadzało jeden kwintet ITA2, tak jak z czytnika taśmy perforowanej. Przełączenie pocztu liter na cyfry, lub odwrotnie, wymagało wprowadzenia kodu przełączającego klawiszem FS, LS, lub tp. Nazwijmy ten tryb pracy klawiatury trybem Baudot.

Klawiatura emulatora ma dwukrotnie więcej klawiszy i potrafi ponadto działać w sposób podobny do klawiatury komputera PC. W tym trybie naciskamy klawisze literowe i cyfrowe, a emulator automatycznie generuje odpowiednie kody przełączające. Jeśli dalekopis ma trzy poczty znaków, np. jeszcze małe litery, to wspomagamy się klawiszami CapsLock i Shift. Czwarty poczet możemy przywołać klawiszem Alt. Klawisze CapsLock, Shift i Alt same z siebie nie wprowadzają żadnych

kodów przełączających. Jeśli emulator stwierdzi, że kolejny znak należy do innego pocztu, to wygeneruje kody przełączające zgodnie z typem dalekopisu. Jedno naciśnięcie może wygenerować serię kodów ITA2, które będą czytane przez kolejne rozkazy czytania.

Tryb PC klawiatury dalekopisu umożliwia pisanie bezpośrednio na klawiaturze komputera PC, zamiast na wirtualnej, wyświetlanej na ekranie. Znak z klawiatury PC, którego nie ma w alfabecie danego typu dalekopisu, zostanie zignorowany, z wyjątkiem znaku tabulacji poziomej Tab. Klawiatura wirtualna umożliwia za to wprowadzanie znaków nieistniejących na klawiaturze PC.

Przełączanie klawiatury z trybu pracy Baudot na tryb PC i odwrotnie następuje automatycznie:

- na tryb PC – w następstwie użycia klawiszy CapsLock, Shift, Alt i NewLine,
- na tryb Baudot – w następstwie użycia klawiszy CR, LF, NU, FS i LS.

Tryb PC klawiatury ma moc przekształcania znaków komputera PC na kody ITA2, a to pozwoliło na podpinanie zwykłego pliku tekstowego w miejsce klawiatury. Plik taki działa jak klawiatura PC z dwoma wyjątkami:

- znak nieznany dalekopisowi jest zamieniany na SP (odstęp), oraz
- w trybie małych liter, jedna lub dwie wielkie litery w pliku tekstowym nie powodują przejścia w tryb wielkich liter, ale ciąg trzech lub więcej kolejnych wielkich liter automatycznie generuje kody przełączające na tryb wielkich liter, tj. symulowane jest wciśnięcie klawisza CapsLock.

Znak tabulacji poziomej Tab, pochodzący zarówno z klawiatury komputera PC, jak i z pliku tekstowego podpiętego w miejsce klawiatury, jest zamieniany na serię od 1 do 8 spacji SP tak, by przesunąć karetkę dalekopisu do najbliższego punktu tabulacji (za wielokrotnością 8 znaków).

Po zatrzymaniu wejścia z pliku tekstowego (co przełącza wejście na klawiaturę) można pozycjonować tekst zaznaczając podwójnym kliknięciem fragment, począwszy od którego ma być kontynuowane czytanie, a następnie z powrotem przełączyć wejście na plik tekstowy. Możliwe jest też przełączanie urządzenia wejściowego na inne między jednym a drugim rozkazem czytania z dalekopisu.

Plik tekstowy pozwala wcześniej przygotować dane czytane z dalekopisu, które w innym razie należałoby wpisywać ręcznie. Dodatkowo mamy naturalny konwerter tekstu Unicode na kod ITA2, a program kopiowania tasiemki perforowanej na drukarkę jest konwerterem kodu ITA2 na Unicode.

UWAGA 1: Pisanie na klawiaturze możliwe jest tylko w odpowiedzi na rozkaz czytania z komputera – jest to rozkaz o kodzie TPG=126, ale urządzenie samoczynnie, miganiem lampki, daje sygnał do pisania.

UWAGA 2: Pisanie na powolnych biegach emulatora lub w trybie pracy start-stopowej może być uciążliwe, ponieważ konieczne są na przemian naciśnięcia przycisku ponaglenia  lub [StartCPU], oraz znaków na klawiaturze, a do tego wciskanie klawiszy odbywa się z długimi przerwami.

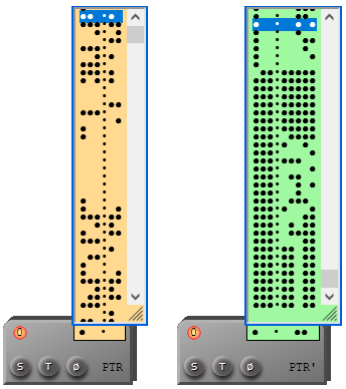
Taśmy perforowane

Taśmę perforowaną 5-kanałową reprezentuje w emulatorze plik binarny .pt5, którego jeden bajt zawiera jeden kwintet kodu ITA2. Kwintet zapisany jest na bitach 7..3 bajtu. Pozostałe bity powinny być wyzerowane; jeśli nie są, to zostaną zignorowane.

Taśmę perforowaną 8-kanałową reprezentuje w emulatorze plik binarny .pt8, którego jeden bajt zawiera jeden oktet. Oktet zapisany jest na bitach 7..0 bajtu.

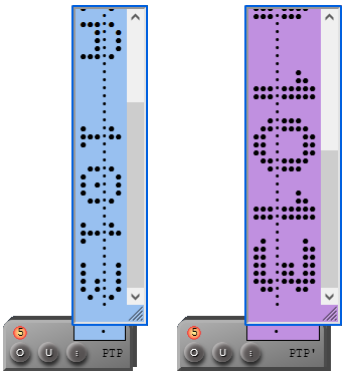
Położenie bitów kwintetu i oktetu w bajcie jest zgodne z opisanym w [2] montowaniem i czytaniem taśmy 5-kanałowej w czytniku 8-kanałowym, oraz taśmy 8-kanałowej w czytniku 5-kanałowym.

Czytnik taśmy perforowanej ma:



- napis PTR lub PTR’ oznaczający czytnik,
- lampkę z numerem adresowym. Numer jest cyfrą ósemkową odpowiadającą pierwszej cyfrze kodu TPG rozkazu czytania. Lampka świeci się w trakcie pracy urządzenia, miganiem wskazuje brak gotowości do pracy,
- prostokątne pole ukazujące wczytany kwintet lub oktet, i służące do otwierania i zamykania podglądu zamontowanej tasiemki,
- przycisk (S) służący do startowania i stopowania pracy urządzenia; montowanie, demontowanie i pozycjonowanie taśmy możliwe jest tylko przy zatrzymanym czytniku. Pozycjonowanie taśmy polega na podwójnym kliknięciu w kwintet lub oktet, który ma być czytany jako następny,
- przycisk (T) służący do montowania taśmy perforowanej,
- przycisk (Ø) służący do demontowania taśmy perforowanej.

Perforator taśmy perforowanej ma:



- napis PTP lub PTP’ oznaczający perforator,
- lampkę z numerem adresowym. Numer jest cyfrą ósemkową odpowiadającą pierwszej cyfrze kodu TPG rozkazu pisania. Lampka świeci się gdy perforowanie jest włączone, a w przeciwnym razie jest zgaszona.
- prostokątne pole ukazujące ostatni wyperforowany kwintet lub oktet, i służące do otwierania i zamykania podglądu perforowanej tasiemki,
- przycisk (O) służący do włączania i wyłączania perforowania. Przy wyłączonym perforowaniu, kody posyłane do perforatora trafiają w nicość (lub ew. do samopisu).
- przycisk (U) służący do odrywania wyperforowanej taśmy w celu jej zarchiwizowania,
- przycisk (⋮) służący do wypuszczenia pustych znaków (NU) – zwykle na początku lub końcu taśmy perforowanej.

Wykresy

Wykresy rysowane na samopisie MAW są tworzone jako czarno-białe obrazy w formacie plików .bmp.

Samopis jest podłączony równolegle z perforatorem 8-kanałowym. Wykres produkowany przez komputer może być wyprowadzony bezpośrednio na samopis, albo na perforator jako taśma, którą następnie można skopiować (tutaj prostym programem) z czytnika na samopis (sposób ten był stosowany w związku z powolnością samopisu).

Kody od 0 do 254 wysyłane do samopisu stawiają punkty w poprzek papieru w pozycjach od 0 do 254 (0 – punkt po lewej stronie, 254 – po prawej). Kod 255 powoduje wysuw papieru z góry na dół. Przycisk (A) przełącza urządzenie w tryb automatycznego wysuwu, w którym po postawieniu punktu następuje samoczynny wysuw papieru. W tym trybie kod 255 stawia kropkę w pozycji 255, co udostępnia łącznie 256 punktów w poprzek papieru (emulator stawia tę kropkę również bez autowysuwu, lecz pozostaje ona poza obszarem rysunku).

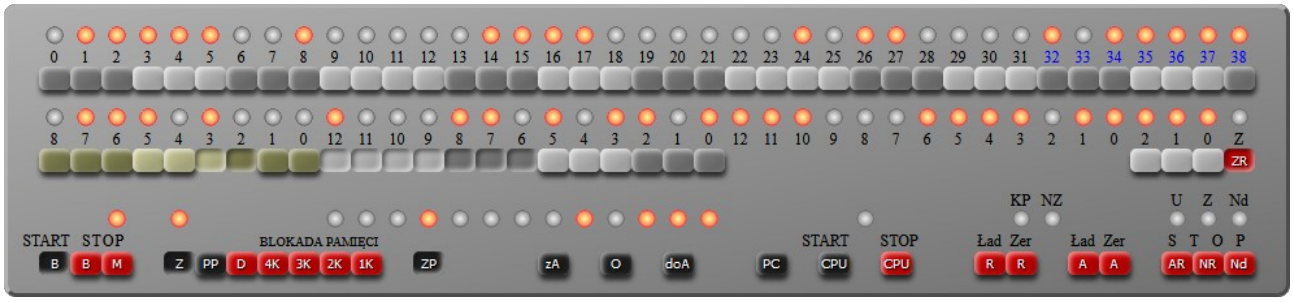
Samopis ma:

- napis MAW’ oznaczający samopis,
- lampkę z numerem adresowym. Numer jest cyfrą ósemkową odpowiadającą pierwszej cyfrze kodu TPG rozkazu pisania. Lampka świeci się gdy rysowanie jest włączone, a w przeciwnym razie jest zgaszona.
- prostokątne animowane pole wyobrażające wykres, służące do otwierania i zamykania podglądu drukowanego wykresu,
- przycisk (O) służący do włączania i wyłączania rysowania. Przy wyłączonym rysowaniu, kody posyłane do samopisu trafiają w nicość, lub na podłączony równolegle perforator.
- przycisk (U) służący do odrywania rysunku w celu jego zarchiwizowania,
- przycisk (A) służący do włączania / wyłączania automatycznego wysuwu papieru.

Obsługa operatorska

Pomimo dużych rozmiarów, ODRA 1003/1013 miała wszelkie cechy komputera osobistego. Uruchamianie, testowanie, a nawet użytkowanie programu wymagało szczegółowej informacji, jakie programy narzędziowe będą potrzebne, kiedy i jakie tasiemki należy montować, na które czytniki, jakie przyciski pulpitu należy wciskać lub zwalniać, itd. Szczególnie uruchamianie i testowanie, zależne od uzyskiwanych wyników, wymagało najczęściej udziału autora programu przy obsłudze maszyny. Dlatego pracę organizowano w formie sesji, w których cała maszyna pozostawała do dyspozycji programisty, z konieczności znającego jej obsługę.

Pulpit



Rząd górny – Akumulator A

Lampki: wyświetlają aktualną zawartość rejestru akumulatora A. Po wciśnięciu przycisku [ZP], ostatnie 7 lampek wskazuje wartość rejestru Ac cechy liczby zmiennoprzecinkowej.

Przyciski: stabilne, służące jako przełączniki – ich stan może być odczytany rozkazem 326, albo załadowany do akumulatora przyciskiem [ŁadA] przy zatrzymanej maszynie.

Rząd środkowy – Rejestr rozkazów R

Lampki: aktualna zawartość rejestru rozkazów R. Od lewej:
– lampki 8-0 – część OR (kod TPG) rozkazu
– lampki 12-0 – część AR (N) rozkazu
– lampki 12-0 – część NR (K) rozkazu
– lampki 2-0 – część MR (B) rozkazu
– lampka nad przyciskiem [ZR] – bit MR (Z) modyfikacji rozkazu

Przyciski: stabilne, do ustawiania rozkazów (przyciskiem [ŁadR] przy zatrzymanej maszynie):
– przyciski 8-0 – do ustawiania części OR (kodu TPG) rozkazu
– przyciski 12-0 – do ustawiania części AR (N) rozkazu
– brak przycisków do ustawiania części NR (K) rozkazu – zbędne
– przyciski 2-0 – do ustawiania części MR (B) rozkazu
– brak przycisku do ustawiania bitu modyfikacji MR (Z) rozkazu – zbędny – w jego miejsce jest przycisk [ZR] Zmiany Reżimu adresacji pamięci
– przycisk [ZR] – Zmiana Reżimu adresacji komórek pamięci: zwolniony – adresacja normalna, wciśnięty – adresacja z przeplotem co 7 komórek.

UWAGA: Zmiana reżimu adresacji pamięci dopiero po wczytaniu programu lub w trakcie wykonywania, grozi jego dezorganizacją.

Rząd dolny – przyciski operacyjne i towarzyszące im lampki sygnalizacyjne

13 lampek rejestru SA – wyświetlają adres ostatnio pobranego rozkazu.

przycisk [StartB] – stabilny. Włącza napięcia niestabilizowane sygnalizowane pierwszą lampką: „Napięcia niestabilizowane”; włącza zasilanie instalacji komputerowej, w tym urządzeń wejścia/wyjścia (przy włączaniu maszyny wciskany jako pierwszy).

przycisk [Z] – stabilny. Włącza napięcia stabilizowane sygnalizowane drugą lampką: „Napięcia stabilizowane”, oraz włącza zasilanie maszyny (przy włączaniu maszyny wciskany jako drugi).

przycisk [PP] – Przygotowanie Pracy Maszyny – niestabilny. Synchronizuje pracę maszyny z obrotami bębna (przy włączaniu maszyny wciskany jako trzeci). Wciśnięcie go powoduje ustawienie bębna w pozycji optymalnej do odszukania komórki ze strefy 0. Nie należy wciskać go podczas wykonywania programu, gdyż może to zdezorganizować pracę maszyny.

przycisk [StopM] – niestabilny. Wyłącza zasilanie maszyny i napięcia stabilizowane – gaśnie lampka „Napięcia stabilizowane” (przy wyłączaniu maszyny wciskany jako pierwszy).

przycisk [StopB] – niestabilny. Wyłącza zasilanie całej instalacji komputerowej – gaśnie lampka „Napięcia stabilizowane” (przy wyłączaniu maszyny wciskany jako drugi).

przycisk [D] – Detekcja niezgodności Komputer-Program – stabilny. Oryginalnie był to przycisk blokujący dalekopis, by nie zakłócał pracy maszyny. W emulatorze ta jego rola stała się zbędna i został zaadaptowany do wykrywania rozkazów zdefiniowanych inaczej w wyższych modelach maszyny, niż w danym: zwolniony – realizuje rozkazy zgodnie z ich definicją w danym modelu maszyny (jako NIC NIE RÓB), wciśnięty – stopuje maszynę po wykryciu rozkazu, który został zdefiniowany inaczej w wyższym modelu. Niezależnie od stanu przycisku, CPU sygnalizuje takie rozkazy objaśnieniem.

lampka KP – Kontrola Komputer-Program – Oryginalnie zapalała się, gdy zatrzymanie maszyny nastąpiło z powodu wykrycia błędu parzystości w pamięci ferrytowej, tj. gdy liczba jedynek w 40-bitowym słowie (39 bitów danych plus bit parzystości) była nieparzysta. W emulatorze ta jej rola stała się zbędna i została zaadaptowana (wraz z przyciskiem [D]) do wykrywania rozkazów zdefiniowanych inaczej w wyższych modelach maszyny, niż w danym.

przyciski [1K], [2K], [3K], [4K] – blokada zapisu do pamięci – stabilne. Każdy blokuje zapis do 1024 komórek o adresach odpowiednio 0000-1023, 1024-2047, 2048-3071 i 3072-4095.

przycisk [ZP] – Zmienny Przecinek – stabilny. Wciśnięty, na ostatnich siedmiu lampkach akumulatora wyświetla zawartość akumulatora cechy Ac.

przycisk [zA] – niestabilny. Przy zatrzymanej maszynie przesyła zawartość akumulatora A do komórki o adresie ustawionym na części AR przycisków rejestru rozkazów (działa jak ustawienie rozkazu TPG=401 i naciśnięcie przycisku [O]).

przycisk [O] – niestabilny. Przy zatrzymanej maszynie powoduje wykonanie operacji ustawionej na przyciskach rejestru rozkazów R.

przycisk [doA] – niestabilny. Przy zatrzymanej maszynie przesyła zawartość komórki o adresie ustawionym na części AR przycisków rejestru rozkazów do A (działa jak ustawienie rozkazu TPG=050 i naciśnięcie przycisku [O]).

przycisk [PC] – stabilny: wciśnięty – Praca Ciągła, zwolniony – Praca Start-Stopowa (wykonanie kolejnego rozkazu wymaga naciśnięcia przycisku [StartCPU], po czym maszyna znowu się zatrzyma).

przyciski [StartCPU] i [StopCPU] (niestabilne) oraz lampka Start – uruchamianie i zatrzymywanie pętli głównej sterowania. Lampka świeci się podczas pracy maszyny, a gaśnie po zastopowaniu.

przycisk [ŁadR] – niestabilny. Przy zatrzymanej maszynie ładuje stan przycisków [OR], [AR] i [MR] do rejestru rozkazów R. Po powtórnym naciśnięciu ładuje stan AR też do NR.

przycisk [ZerR] – niestabilny. Przy zatrzymanej maszynie powoduje wyzerowanie wszystkich bitów rejestru rozkazów R.

przycisk [ŁadA] – niestabilny. Przy zatrzymanej maszynie przesyła stan przycisków akumulatora do rejestru A (do akumulatora), ale bez ustawiania rejestrów warunku.

przycisk [ZerA] – niestabilny. Przy zatrzymanej maszynie powoduje wyzerowanie wszystkich bitów akumulatora A, ale bez ustawiania rejestrów warunku.

lampka NZ – Nadmiar Zmiennoprzecinkowy – świeci się, gdy maszyna zatrzymała się z powodu nadmiaru zmiennoprzecinkowego.

lampki U, Z, Nd – Ujemne, Zero, Nadmiar stałoprzecinkowy – wskazują stan rejestrów warunków. Świecą się odpowiednio przy warunku Ujemne, Zero, Nadmiar stałoprzecinkowy.

przycisk [StopAR] – stabilny. Przy wciśniętym przycisku maszyna zatrzyma się po wykonaniu rozkazu mającego skuteczny (po ewentualnej B-modyfikacji) adres AR równy ustawionemu wcześniej na przyciskach części AR rejestru rozkazów.

przycisk [StopNR] – stabilny. Przy wciśniętym przycisku maszyna zatrzyma się po wykonaniu rozkazu mającego skuteczny (po ewentualnej B-modyfikacji) adres NR równy ustawionemu wcześniej na przyciskach części AR rejestru rozkazów.

przycisk [StopNd] – stabilny. Przy wciśniętym przycisku maszyna zatrzyma się po wystąpieniu nadmiaru stałoprzecinkowego.

Włączanie maszyny

- włączyć przyciskiem [StartB] zasilanie całej instalacji komputerowej
- włączyć przyciskiem [Z] maszynę
- wcisnąć przycisk [PP] (w emulatorze przydatne tylko do resetowania pozycji bębna)

Wyłączanie maszyny

- wcisnąć przycisk [StopCPU] i odczekać do zatrzymania się CPU (aż zgaśnie lampka CPU)
- wcisnąć przycisk [StopM] i odczekać do zgaśnięcia lampki „Napięcia stabilizowane”
- wcisnąć przycisk [StopB]

Emulator pozwala wyłączyć maszynę lub całą instalację bezpośrednio przyciskiem [StopM] lub [StopB], ale i tak zostaje zachowana właściwa sekwencja wyłączeń. Zatrzymanie CPU następuje wyłącznie po zakończeniu wykonywania się rozkazu, więc na powolnych biegach mogą być konieczne ponaglenia lub przyspieszenie emulatora.

Ładowanie z pulpitu rejestru akumulatora A

- ustawić przyciski akumulatora: wciśnięty = 1, zwolniony = 0
  Samo wciśnięcie przycisków jeszcze nie ładuje rejestru akumulatora A, ale istnieje rozkaz odczytu stanu przycisków, wykorzystywanych do ustawiania opcji programów
- nacisnąć przycisk [ŁadA] – wartość ustawiona przyciskami przeniesie się do rejestru A i wyświetli na lampkach, przy czym warunki U, Z i Nd pozostaną niezmienione
- wygodny przycisk [ZerA] zeruje akumulator

Przyciski akumulatora można wciskać w dowolnej chwili, natomiast przyciski [ŁadA] i [ZerA] działają tylko przy włączonej, ale zatrzymanej maszynie. Korzysta się z nich do ingerowania w uruchamiany lub testowany program, albo do wprowadzania słów do pamięci, np. krótkich programów.

Ładowanie z pulpitu rejestru rozkazów R

- ustawić przyciski rejestru rozkazów: wciśnięty = 1, zwolniony = 0
  Samo wciśnięcie przycisków jeszcze nie ładuje rejestru, ale wyznacza adres komórki której treść ma być przesłana z/do akumulatora przyciskami odpowiednio [zA] i [doA], lub adres stopu selektywnego żądanego przyciskami [StopAR] i/lub [StopNR], albo wyznacza rozkaz jaki ma zostać wykonany przyciskiem [O]
- nacisnąć przycisk [ŁadR] – wartość ustawiona przyciskami przeniesie się do rejestru rozkazów i wyświetli na lampkach, przy czym druga część adresowa NR (K) i bit modyfikacji MR (Z) rozkazu pozostaną niezmienione; zwykle nie trzeba ustawiać z pulpitu drugiej części adresowej rejestru R, ale da się to zrobić – powtórne naciśnięcie [ŁadR] skopiuje pierwszą część adresową AR (N) na drugą część adresową
- wygodny przycisk [ZerR] zeruje rejestr rozkazów

Przyciski rejestru rozkazów można wciskać w dowolnej chwili, natomiast [ŁadR] i [ZerR] działają tylko przy włączonej, ale zatrzymanej maszynie.

Rejestr rozkazów ładuje się głównie w celu ustawienia adresu startowego uruchamianego programu.

Uruchamianie programu znajdującego się w pamięci

- jeśli przed uruchomieniem programu trzeba wykonać z pulpitu operacje przyciskami [zA], [doA] lub [O], to należy je wykonać przed ustawieniem w rejestrze rozkazów adresu programu
- ustawić w rejestrze rozkazów adres programu (adres komórki pamięci zawierającej pierwszy rozkaz do wykonania). Adres ten powinien być podany w opisie programu. Ustawić przyciski części AR (N) rejestru rozkazów i wcisnąć przycisk [ŁadR]
- jeśli to konieczne do testowania programu, ustawić przyciski rejestru rozkazów na adres selektywnego stopu żądanego przyciskami [StopAR] i/lub [StopNR]. Adres stopu można zmieniać w każdej chwili, najlepiej zatrzymując na ten moment maszynę przyciskiem [StopCPU]
- ustawić inne przyciski pulpitu, np. akumulatora, zgodnie z opisem programu
- stosownie do bieżących potrzeb włączyć inne opcje przyciskami [PC], [StopAR], [StopNR], [StopNd], [1K], [2K], [3K], [4K], oraz [D] – przyciski te można włączać i wyłączać w każdej chwili
- nacisnąć przycisk [StartCPU] – zostanie wykonany pierwszy rozkaz, a po nim kolejne. Maszyna zatrzyma się gdy np. zwolniony jest przycisk [PC], wciśnięty [StopAR], [StopNR] lub [StopNd], wystąpił nadmiar zmiennoprzecinkowy, nastąpiło dzielenie przez zero, włączona detekcja [D] niezgodności Komputer-Program, naciśnięto [StopCPU], albo wykonał się rozkaz Stopu. Przycisk [StartCPU] wznawia pracę.

Wczytywanie programu do pamięci programem STAŁYM

Program STAŁY WPROWADZAJĄCY znajduje się w pamięci stałej pod adresem $17700_{(8)}$ i działa w obu reżimach adresacji: normalnym i przeplotowym. Program służy do wczytywania do pamięci programów/danych zapisanych na tasiemce perforowanej 5-kanałowej w kodzie PIĄTKOWYM. Pełny opis programu STAŁEGO znajduje się wśród instrukcji emulatora.

W celu wczytania programu w kodzie PIĄTKOWYM należy:

- ustawić przyciskiem [ZR] reżim adresacji pamięci właściwy dla wczytywanego programu – informacja ta powinna być podana w opisie programu
- założyć taśmę perforowaną z programem w kodzie PIĄTKOWYM do czytnika nr 0
- na pulpicie maszyny ustawić wartość $17700_{(8)}$ przyciskami do ustawiania części AR (N) rejestru rozkazów (właśnie w takim ustawieniu pokazane są na ilustracji)
- nacisnąć przycisk [ŁadR], co ustawi adres $17700_{(8)}$ na części AR (N) rejestru rozkazów R
- wcisnąć przycisk [PC]
- nacisnąć przycisk [StartCPU], co uruchomi wczytywanie kodu z tasiemki do pamięci

Po wczytaniu taśmy na lampkach akumulatora wyświetli się różnica symetryczna między sumą kontrolną obliczoną podczas wczytywania, a zapisaną na taśmie:

- jeśli lampki akumulatora są zgaszone – stan poprawny
- jeśli nie wszystkie lampki akumulatora są zgaszone – sprawdzić, czy nie jest tak z powodu wciśniętego przycisku [ZP]. Jeśli nie, to stan lampek może wynikać z braku lub błędnej sumy kontrolnej na taśmie. W przeciwnym razie nastąpiło przekłamanie lub niepoprawna obsługa urządzeń i należy powtórzyć wczytywanie.

Program STAŁY WPROWADZAJĄCY zatrzymuje się na rozkazie :726 00000 17700 00 – aby wczytać następne bloki danych wystarczy po założeniu taśmy nacisnąć przycisk [StartCPU].

Wczytywanie, uruchamianie, testowanie i eksploatacja programów zapisanych w postaci innej niż kod PIĄTKOWY wymaga wczytania odpowiedniego programu czytającego, translatora, autokodu, itp., i musi odbywać się zgodnie z jego instrukcją. Takim programem jest język PODSTAWOWY (PJZ – Podstawowy Język Zewnętrzny) służący do wczytywania programów źródłowych w języku programowania na najniższym poziomie maszynowym, generowania kodu wykonywalnego i uruchamiania go, oraz zawierający pakiet najbardziej potrzebnych podprogramów. Wśród innych dostępnych na tę maszynę były: JAS (Język Adresów Symbolicznych), autokody MOST i FALA, symulator TYMAC, czy system operacyjny PROM.

Testowanie programu

Dostęp do pulpitu maszyny umożliwia testowanie programu na wiele sposobów. Najbardziej podstawowy to uruchomienie programu przy zwolnionym przycisku [PC] i rozkaz po rozkazie dokonywanie inspekcji zawartości rejestru rozkazów, akumulatora, innych rejestrów i pamięci.

Przyciski [zA] i [doA] umożliwiają przesyłanie treści między rejestrem A a pamięcią. Dzięki nim można na bieżąco korygować rozkazy lub dane w pamięci (odnotowując te poprawki).

Przyciskiem [O] możemy dokonywać bardziej złożonych ingerencji w program, wykonując dodatkowe rozkazy korygujące.

Jeśli chcemy zatrzymać program w dalszej części, możemy ustawić jej adres przyciskami części AR (N) rejestru rozkazów, wcisnąć [StopNR] oraz ustawić pracę ciągłą [PC] – program zatrzyma się po wykonaniu rozkazu z tak wskazaną skuteczną drugą częścią adresową.

Przycisk [StopAR] zatrzyma program po wykonaniu rozkazu mającego skuteczną pierwszą część adresową wskazaną przyciskami części AR (N) rejestru rozkazów. Pierwsza część adresowa to bezpośredni argument N, adres argumentu (komórki pamięci, zmiennej), adres skoku warunkowego lub adres podprogramu. W przypadku podprogramu zatrzymanie nastąpi zarówno przy wejściu do podprogramu jak i przy wyjściu po śladzie.

Z kolei przycisk [StopNd] pozwoli wyłapać miejsce powstania nadmiaru stałoprzecinkowego.

Wystąpienie nadmiaru zmiennoprzecinkowego zawsze zatrzymuje maszynę zapalając lampkę NZ.

Wstrzymanie pracy maszyny na operacji czytania można spowodować zatrzymaniem właściwego czytnika lub tekstu zastępującego klawiaturę dalekopisu.

Emulator oferuje niezależne od maszyny wygody weryfikowania poprawności działania programu: poprzez spowalnianie biegu i ułatwiony wgląd w rejestry i pamięć.

Dodatkowym narzędziem emulatora jest wykrywanie rozkazów, które z powodu uruchomienia programu na niewłaściwym modelu maszyny być może wykonują się jako „NIC NIE RÓB” zamiast wymaganej funkcji. Służy do tego zaadaptowany przycisk [D] wraz z lampką KP.

Przykłady obsługi operatorskiej

Przykład 1 – perforowanie kodu wizualnego

Załóżmy, że chcemy na taśmie wyperforować kod wizualny o treści „▒WR-Z-12▒03-IV-113▒”, poprzedzony i zakończony rozbiegówkami (patrz dalej). Posłużymy się w tym celu programem „PL2 5Hs.pgm.pt5” – jednym z kilku gotowych w postaci taśmy w kodzie PIĄTKOWYM:

1. Uruchamiamy emulator. Zmieniamy typ dalekopisu na TTY MKD-2 PL2, ponieważ nasz program jest zgodny z alfabetem tego dalekopisu (przy pomocy innych dalekopisów, np. TTY MKD-2 PL3 lub TTY MKD-2 PL4 również możemy uzyskać właściwy wynik, ale wymagałoby to więcej uwagi). Robimy to klikając w plakietkę z nazwą typu.
2. Włączamy zasilanie przyciskiem [StartB], oraz włączamy maszynę przyciskiem [Z] – nasz program nie wymaga szczególnego modelu komputera.
3. Serduszkami emulatora możemy zajrzeć do trzewi CPU i zakamarków pamięci. Zobaczymy, że na ostatniej ścieżce pamięci bębnowej coś jest – to program STAŁY.
4. Musimy wczytać nasz program do pamięci. Program jest na taśmie 5-kanałowej, w postaci binarnej, w tzw. kodzie PIĄTKOWYM. Program STAŁY potrafi wczytywać taśmy w tym kodzie:
   a) Na pulpicie maszyny, szarymi przyciskami AR (N) rejestru rozkazów (środkowy rząd), ustawiamy adres programu, jaki chcemy uruchomić – a chcemy uruchomić program STAŁY WPROWADZAJĄCY, znajdujący się w pamięci pod adresem $17700_{(8)}$. Zatem wciskamy przyciski od 12 do 6, pozostawiając zwolnione przyciski od 5 do 0.
   b) Naciskamy przycisk [ŁadR], aby adres ustawiony przyciskami AR (N) rejestru rozkazów wprowadzić do rejestru rozkazów – lampki nad przyciskami wskazują adres znajdujący się w rejestrze rozkazów.
   c) Jeśli program ma się wykonywać bez zatrzymań, to należy wcisnąć przycisk [PC], oraz ustawić co najmniej 4. bieg emulatora.
   d) Uruchamiamy program STAŁY WPROWADZAJĄCY naciskając przycisk [StartCPU]. Po krótszej lub dłuższej chwili (zależnie od biegu), program WPROWADZAJĄCY zażąda tasiemki do wczytania, co czytnik nr 0 zasygnalizuje miganiem lampki.
   e) Zakładamy do czytnika taśmę z programem perforującym kod wizualny: przyciskiem (T) wybieramy taśmę „PL2 5Hs.pgm.pt5”, a następnie przyciskiem (S) startujemy czytnik, pozwalając na czytanie taśmy.
   f) Po wczytaniu programu lampka „Start-Stop CPU” gaśnie. Sprawdzamy stan lampek akumulatora – zgaszone oznaczają poprawne wczytanie. Przyciskiem (S) wyłączamy czytnik, a przyciskiem (Ø) zdejmujemy taśmę wczytanego programu.
5. Program perforujący został wczytany do pamięci maszyny pod adres $17400_{(8)}$ – uruchamiamy go:
   a) Ładujemy do rejestru rozkazów adres $17400_{(8)}$ początku programu perforującego – analogicznie jak w punktach 4a) i 4b).
   b) Przyciskiem (⋮) na perforatorze PTP5 wypuszczamy nieco pustej taśmy jako wstępną rozbiegówkę i naciskamy przycisk [StartCPU] – dalekopis miganiem lampki zachęca do pisania.
   c) Piszemy wymagany tekst „▒WR-Z-12▒03-IV-113▒” – całkowicie zadziurkowany znak „▒” uzyskamy wprowadzając kod CR. Kod LF oznacza koniec naszego tekstu – program zakończy pracę (i będzie gotowy do ponownego wystartowania), a my wypuszczamy przyciskiem (⋮) nieco pustej taśmy jako rozbiegówkę właściwą.

Jest to sposób opatrzenia taśmy w kod wizualny i niezbędne rozbiegówki, na którą następnie możemy wyprowadzić jakieś swoje dane lub program w kodzie PIĄTKOWYM.

Przykład 2 – rysowanie wykresu

Załóżmy, że mamy taśmę z wyperforowanym wykresem funkcji jednej zmiennej. Taka taśma mogła powstać jako wynik programu rysującego wykres, ale z uwagi na powolność urządzenia rysującego, wyjście zostało skierowane na perforator taśmy 8-kanałowej. Każdy oktet na taśmie to kod sterujący pracą samopisu. Rysunek wykresu możemy uzyskać uruchamiając program przepisujący kody z taśmy na samopis MAW. Program taki znajduje się w pamięci stałej ODRY UMCS. A więc do pracy:

1. Uruchamiamy emulator i włączamy zasilanie przyciskiem [StartB].
2. Samopis MAW jest urządzeniem 8-kanałowym, zatem musimy mieć ODRĘ UMCS, bo tylko taka jest odpowiednia – klikamy w emblemat dotąd, aż model komputera zmieni się na ODRA UMCS. Teraz włączamy maszynę przyciskiem [Z].
3. Serduszkami emulatora możemy zajrzeć do trzewi CPU i zakamarków pamięci. Zobaczymy, że na ostatniej ścieżce pamięci bębnowej coś jest – to program STAŁY.
4. Musimy uruchomić program kopiujący tasiemkę na samopis:
   a) Przygotowujemy samopis: jest on podłączony równolegle z perforatorem 8-kanałowym, przy czym normalnie perforator jest włączony, a samopis nie. Przyciskiem (O) wyłączamy perforator (nie potrzebujemy kopii naszej tasiemki) i takim samym przyciskiem włączamy samopis MAW. Zapalona lampka wskazuje, że samopis jest włączony. Jeśli przy tym chcemy obserwować proces drukowania wykresu, to klikamy w prostokąt wyobrażający wykres, co ukaże nam papier wydruku.
   b) Normalnie samopis po otrzymaniu kodu 255 wysuwa papier o 1 linię pikseli. Jeśli jednak drukujemy wykres jednej funkcji, możemy kodu tego używać do stawiania punktu w pozycji 255, polegając na samoczynnym wysuwie papieru. Samopis działa w ten sposób po wciśnięciu przycisku (A). Wciskamy go, bo chociaż to rzadkość, to nasz wykres tego wymaga.
   c) Na pulpicie maszyny, szarymi przyciskami AR (N) rejestru rozkazów (środkowy rząd), ustawiamy adres programu, jaki chcemy uruchomić – a chcemy uruchomić program STAŁY DRUKOWANIE WYKRESÓW, znajdujący się w pamięci stałej pod adresem $17657_{(8)}$.
   d) Naciskamy przycisk [ŁadR], aby adres ustawiony przyciskami AR (N) rejestru rozkazów wprowadzić do rejestru rozkazów – lampki nad przyciskami wskazują adres znajdujący się w rejestrze rozkazów.
   e) Jeśli program ma się wykonywać bez zatrzymań, to należy wcisnąć przycisk [PC], oraz ustawić co najmniej 4. bieg emulatora.
   f) Uruchamiamy program DRUKOWANIA WYKRESÓW naciskając przycisk [StartCPU]. Po chwili program zażąda taśmy z danymi do skopiowania na samopis – miga lampka czytnika 8-kanałowego PTR2’.

g) Zakładamy przykładową taśmę „wykres z autowysuwem.pt8” do czytnika. Oczywiście taśmę można było zamontować też wcześniej.

h) Po wystartowaniu czytnika program kopiuje dane na samopis, na którym pojawia się rysunek. Ten program działa w nieskończonej pętli – nie wie kiedy jest koniec danych. Po przeczytaniu całej taśmy lampka czytnika znowu zaczyna migać w oczekiwaniu na dalsze dane. Możemy założyć następną tasiemkę, albo zignorować to, a wydruk oderwać przyciskiem (U).

i) Niezależnie od tego co zrobimy, widzimy, że maszyna wykonuje rozkaz czytania czekając na urządzenie i lampka „Start-Stop CPU” nie gaśnie. Jedynym sposobem zatrzymania programu jest: nacisnąć przycisk [StopCPU], kliknąć w któryś oktet tasiemki wskazując na niej nową pozycję i nacisnąć przycisk (S) na czytniku, by pozwolić rozkazowi wykonać się do końca. Następne rozkazy już się nie wykonają, maszyna zatrzyma się i nasz wydruk, jeśli jeszcze nie został oderwany, nie zostanie naruszony. Takie działanie jest konsekwencją faktu, że maszyna zawsze zatrzymuje się dopiero po wykonaniu rozkazu.

Przygotowywanie taśm perforowanych

Rzeczywisty czytnik taśmy perforowanej to urządzenie, do którego montowanie taśmy wymaga podłożenia początkowego jej fragmentu pod głowicę czytającą, a więc ułożenia taśmy w czytniku tak, by jej część wystawała poza czytnik. Istotne jest przy tym, by nie pominąć danych znajdujących się na początku taśmy, jak też by początkowy odcinek taśmy nie został mylnie podany jako dane. Dla uniknięcia pomyłek przyjęły się pewne sposoby znakowania taśm.

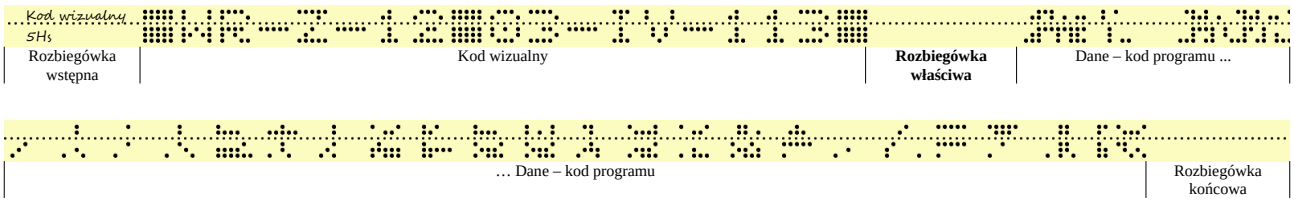
Zaznaczmy z góry, że montowanie taśm w rzeczywistym czytniku wymaga uwagi, natomiast w emulatorze jest znacznie ułatwione. Taśmy są plikami identyfikowanymi stosownie do systemu operacyjnego komputera-hosta i zabiegi opisane poniżej zwykle są zbędne.

Rozbiegówki i kod wizualny

Typowa taśma perforowana zawiera puste odcinki na swoim początku i końcu. Ich zadaniem jest umożliwienie zakładania taśmy do urządzeń, oraz zabezpieczenie początku i końca danych przed uszkodzeniami mechanicznymi.

Opisana wcześniej taśma perforowana z danymi do rysowania wykresu ma tę specyficzną cechę, że każdy oktet taśmy może być potraktowany jako kod sterujący pracą samopisu, w tym oktety całkowicie puste. Oznacza to, że wykres powstający wskutek kopiowania takiej taśmy na samopis zależy od punktu początkowego na taśmie, podłożonego pod głowicę – puste oktety na początku będą traktowane jako kody nakazujące stawianie punktów w pozycji 0 (po lewej stronie papieru).

Odcinki początkowy i końcowy typowej taśmy perforowanej



Typowa taśma perforowana zawiera:

1. rozbiegówkę wstępną, na której można było ręcznie napisać identyfikację taśmy (co ułatwiało odróżnienie początku taśmy od końca, oraz lewej strony od prawej),
2. kod wizualny nanoszony programowo w postaci odpowiedniej perforacji – jest to dodatkowy sposób identyfikacji taśmy przez osobę obsługującą,
3. rozbiegówkę właściwą, pozwalającą wygodnie ułożyć taśmę w czytniku. To ten fragment taśmy podkłada się pod głowicę podczas montowania w czytniku, pomijając kod wizualny. W przypadku rezygnacji z kodu wizualnego mamy do czynienia z jedną rozbiegówką,
4. właściwe dane umieszczone na taśmie: bądź to w kodzie binarnym (np. PIĄTKOWYM), bądź w kodzie tekstowym ITA2,
5. rozbiegówkę końcową.

Początkowy fragment taśmy stanowi tzw. rozbiegówkę i nie powinien zawierać danych. Rozbiegówkę łatwo uzyskać przyciskiem (⋮) perforatora lub klawiszem NU klawiatury dalekopisu. Kod wizualny trudniej, bo np. jednym z kilku przygotowanych programów (ale pracujących w pamięci ferrytowej, co przeszkadza gdy mamy tam już jakieś inne dane lub program).

W przypadku emulatora, rozbiegówki i kod wizualny są zbędne i mogą tylko zakłócić montowanie taśmy, gdyż nie są one przeznaczone do czytania przez programy. Jeśli decydujemy się je umieścić, to należy zadbać, by rozbiegówka i kod wizualny nie utrudniały pozycjonowania taśmy w czytniku.

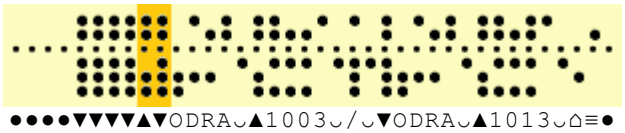
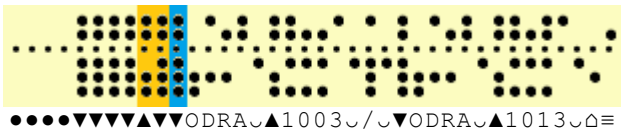
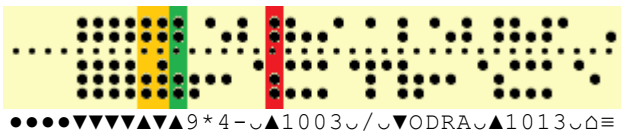
Taśma z danymi wykresu wymaga bardzo uważnego montowania w rzeczywistym czytniku, na szczęście w emulatorze rozbiegówki i kod wizualny, jako zbędne, można pominąć.

Ustalenie trybu pracy i pocztu znaków na taśmie z tekstem

Szczególnej uwagi wymaga taśma, na której wyperforowano tekst w kodzie ITA2. Dalekopis bezpośrednio po włączeniu jest w takim stanie, że pierwszy posłany doń zwykły kod (tj. różny od NU – null, FS – figure shift i LS – letter shift) jest interpretowany wg pocztu liter w trybie wielkich liter. Jednakże podczas pracy stan dalekopisu zmienia się. Należy uwzględniać fakt, że dalekopis zwykle pozostaje w stanie pozostawionym po wcześniejszej pracy. Dlatego taśmy z tekstem przygotowanym do oddrukowania na dalekopisie, jak też jako dane do programów (np. teksty źródłowe programów), powinny zawierać na początku kody ustalające odpowiedni stan dalekopisu, czyli wybór trybu pracy (tryb wielkich liter, lub tryb małych liter) oraz pocztu znakowego (poczet wielkich liter, znaków figurowych, małych liter, lub znaków specjalnych). Dzięki temu programy czytające tekst odpowiednio zinterpretują kody ITA2.

Mówimy tu skrótowo o „stanie dalekopisu”, chociaż dane mogą być przeznaczone wyłącznie do czytania przez programy. Pamiętajmy jednak, że programy powinny interpretować kod zgodnie ze sposobem interpretacji przez dalekopis, by uniknąć rozbieżności.

Przykładowe sekwencje kodów ustalających stan dalekopisu

Perforacja	*Wydruki na dalekopisach o różnych liczbach pocztów znakowych*	
 ●●●●▼▼▼▼▲▼ODRA␣▲1003␣/␣▼ODRA␣▲1013␣⌂≡●	ODRA 1003 / ODRA 1013	*PL2 – 2 poczty*
	ODRA 1003 / ODRA 1013	*PL3 – 3 poczty*
	ODRA 1003 / ODRA 1013	*PL4 – 4 poczty*
 ●●●●▼▼▼▼▲▼▼ODRA␣▲1003␣/␣▼ODRA␣▲1013␣⌂≡	ODRA 1003 / ODRA 1013	*PL2 – 2 poczty*
	odra 1003 / odra 1013	*PL3 – 3 poczty*
	odra 1003 / odra 1013	*PL4 – 4 poczty*
 ●●●●▼▼▼▼▲▼▲9*4-␣▲1003␣/␣▼ODRA␣▲1013␣⌂≡	9*4- 1003 / ODRA 1013	*PL2 – 2 poczty*
	9*4- 1003 / ODRA 1013	*PL3 – 3 poczty*
	9*4- Ą003 / odra 1013	*PL4 – 4 poczty*

Na początku taśmy z tekstem zwykle umieszcza się kilka kwintetów LS, przełączających na poczet liter. Nie jest jednak pewne, czy na litery wielkie, czy małe.

Kluczowa jest tu nieprzerwana sekwencja dwóch kwintetów FS i LS zaznaczonych kolorem pomarańczowym. Na dalekopisach z dwoma pocztami znaków jest ona obojętna (przełącza na cyfry i z powrotem na litery), a na tych z większą liczbą pocztów resetuje stan dalekopisu do pocztu wielkich liter w trybie wielkich liter.

Dodatkowy kwintet LS, zaznaczony kolorem niebieskim, jest zbędny na dalekopisie z dwoma pocztami znaków, a na tych z większą liczbą pocztów przełącza dalekopis na poczet małych liter (w trybie małych liter). Dalekopis z dwoma pocztami, nie mający małych liter, drukuje je jako wielkie litery.

Dodatkowy kwintet FS, zaznaczony kolorem zielonym, przełącza dalekopis na poczet znaków cyfrowych (figurowych) – rzeczywiście, następne kody (takie same jak w poprzednich przykładach) są traktowane jako znaki figurowe, a nie literowe. Kwintet zaznaczony kolorem czerwonym jest obojętny na dalekopisach z dwoma lub trzema pocztami (bo wtedy przełącza z cyfr na cyfry), ale na dalekopisie wyposażonym w czwarty poczet (dodatkowych znaków specjalnych) przełącza dalekopis na poczet znaków specjalnych na czas jednego znaku, po czym powraca do pocztu cyfr w trybie małych liter.

Przykład czerwonego kwintetu pokazuje nadto, że nie należy nadużywać kodów przełączających, gdyż nie są one obojętne dla urządzeń (dalekopisów lub programów interpretujących) z większą liczbą pocztów znakowych. Dotyczy to również kodu LS przełączającego na litery (chociaż skutkiem może być wydrukowanie wielkiej litery jako małej, lub odwrotnie), oraz sekwencji FS+LS przełączającej dalekopis na poczet wielkich liter w trybie wielkich liter (w trybie, w którym dalekopis operuje tylko dwoma pocztami znaków).

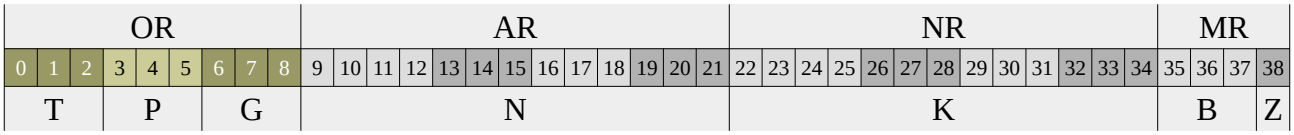
Problemem jest też zaznaczanie końca danych na taśmie. Pozostawia się to do rozstrzygnięcia przy projektowaniu poszczególnych programów i ich danych.

Rozkazy ODRY 1003/1013

Budowa rozkazu

Do zapisu rozkazów stosuje się notację z języka PODSTAWOWEGO – języka programowania na najniższym poziomie maszynowym – gdzie poszczególne cyfry ósemkowe odpowiadają formatowi słowa rozkazowego:

```
          :TPG NNNNN KKKKK BZ
np.       :050 10001 17207 00
```



Rozkaz maszynowy zawiera następujące części składowe:

OR część operacyjna – kod operacji, na który składają się trzy triady: T (typ), P (podstawienie) i G (grupa)

AR pierwsza część adresowa – adres argumentu lub sam argument N, w rozkazach skoku adres następnego rozkazu przy spełnionym warunku, a w niektórych rozkazach wartość nieistotna

NR druga część adresowa – adres K następnego rozkazu do wykonania, ignorowany w rozkazie wysyłającym wynik do rejestru rozkazów (wtedy następny rozkaz wskazuje część NR rozkazu wysłanego do rejestru R)

MR część modyfikacyjna – triada B zawiera numer rejestru B użytego do B-modyfikacji rozkazu lub zawierającego argument rozkazu (jedno nie wyklucza drugiego), a bit Z ustawiony na 1 wskazuje, że rozkaz ma być zmodyfikowany zawartością rejestru B

W wykazie rozkazów zastosowano symboliczny zapis funkcji rozkazów podobny do instrukcji języka JavaScript lub C, a w nim następujące oznaczenia:

x	wartość wchodząca na drugie wejście sumatora
s	wartość wynikowa z sumatora
A	zawartość rejestru akumulatora
B	zawartość rejestru B, tj. B0, B1, B2, B3, B4, B5, B6 lub B7 (w rozkazie SkBG chodzi też o nr rejestru)
M	zawartość rejestru B7
R	rejestr rozkazów, do którego pobrana lub posłana wartość staje się rozkazem do wykonania
AM	zawartość pary rejestrów A i M (tj. bity 0..38 rejestru A, oraz bity 1..38 rejestru M=B7)
AC	zawartość pary rejestrów: Am=A (mantysa), oraz Ac (cecha) liczby zmiennoprzecinkowej
N	wartość pola N rozkazu (argument bezpośredni)
[N]	zawartość komórki pamięci o adresie N
Ω	wskazanie, że ustawiana jest wartość rejestru zaokrągleń
#	wskazanie, że po wykonaniu rozkazu pozostanie pewna wartość w rejestrze roboczym
uzv	wskazanie ustawiania kodu warunku: u – ujemne, z – zero, v – nadmiar stałoprzecinkowy: mała litera – dany warunek jest ustawiany, . – że ten warunek nie jest ustawiany, duża litera – że wystąpi dany warunek, 0 – że ten warunek nie wystąpi
uzvn	jw. a ponadto drugie wskazanie ustawiania nadmiaru: n – może wystąpić nadmiar w fazie mnożenia lub dzielenia stałoprzecinkowego, / – może wystąpić dzielenie przez zero, ● – nastąpi Stop, * – może wystąpić nadmiar zmiennoprzecinkowy
HAK	wskazanie, że rozkaz nie działa dokładnie tak, jak wynikałoby to z kodu operacji, co może wprowadzić w błąd

Wykonanie rozkazu

Wykonanie rozkazu składa się z czterech etapów, wśród których zwróćmy uwagę na etap b – pobranie rozkazu z pamięci. Polega ono na wczytywaniu kolejnych bitów słowa rozkazowego, poczynając od bitu najmłodszego, decydującego o B-modyfikacji. Jeżeli wczytywane słowo należy poddać B-modyfikacji, to zawartość rejestru wskazanego przez triadę B rozkazu jest dodawana do rozkazu w trakcie jego wczytywania. Wszelkie dalsze działania procesora odbywają się na zmodyfikowanym słowie. Wartości TPG, N, K, B i Z, przywoływane w opisach działania rozkazów, oznaczają wartości po wykonanej B-modyfikacji. Np.:

jeżeli w rejestrze B6 znajduje się liczba $01000_{(8)}$, to rozkaz : `032 02000 00000 61`
zmieni się w trakcie B-modyfikacji na rozkaz : `032 02000 01000 61`
czyli rejestr B6 zostanie załadowany nową wartością, ale adres następnego rozkazu do wykonania zależy od starej zawartości B6.

Liczby (adresy, argumenty bezpośrednie, słowa stałoprzecinkowe) są reprezentowane w dwójkowym systemie pozycyjnym, w kodzie uzupełnień do 2. Operacje arytmetyczne: zmiana znaku, wartość bezwzględna, różnica, różnica odwrotna i suma, są zgodne z powszechnie znanymi. Operacje logiczne: koniunkcja i różnica symetryczna, operują równolegle na wszystkich bitach argumentów.

Mantysa i cecha liczby zmiennoprzecinkowej również są reprezentowane w dwójkowym systemie pozycyjnym, w kodzie uzupełnień do 2, czyli inaczej niż np. w standardzie IEEE 754.

Ustawianie warunków

Nie ustawiają warunków rozkazy: niezdefiniowane (Nic), skoków warunkowych (SkZ, SkU, SkD, SkV, SkBG), skoków z licznikiem (SkLC), skoku ze śladem (SkS), zatrzymania CPU (Stop) i przesyłań blokowych (BF61, BF62, FB61, FB62).

Rozkazy sumowań stałoprzecinkowych (grupy G=0,1,2,3), oraz faza sumowania rozkazów mnożenia i dzielenia stałoprzecinkowego (grupy G=4,5) – ustawiają warunki wg stanu sumatora, tj. stosownie do bitów przeniesień z pozycji bitowej nr 0 i z pozycji bitowej nr 1 sumatora. Oznacza to, że nadmiar przy dodawaniu liczb dodatnich ustawia warunek Nd, a zeruje U i Z, zaś nadmiar przy dodawaniu liczb ujemnych ustawia warunki Nd i U, a zeruje Z. Analogicznie dzieje się przy odejmowaniu. Jeśli wynik poślemy np. do akumulatora A, a następnie zawartość A sprawdzimy jakimś porównaniem, to warunki będą odmienne, bowiem żaden inny poza sumatorem rejestr czy komórka pamięci nie ma bitów przeniesień z pozycji bitowych nr 0 i nr 1.

Inne rozkazy ustawiają warunki wg stanu rejestru wynikowego:

- wg rejestru Am (czyli A) zawierającego mantysę liczby zmiennoprzecinkowej,
- wg rejestru AM zawierającego końcowy wynik mnożenia stałoprzecinkowego,
- wg rejestru A – wyniku (końcowego) dzielenia stałoprzecinkowego, dzielenia długiego, przesunięcia, wejścia/wyjścia, czytania klawiatury, zaokrąglenia normalnego.

Oznacza to, że nadmiarowy wynik powoduje ustawienie warunku Nd, bit znaku rejestru równy 1 ustawienie warunku U, a wszystkie bity zerowe ustawienie warunku Z. Czyli np. nadmiar przy przesunięciu w lewo o 1 pozycję liczby dodatniej może ustawić warunki Nd i U, albo Nd i Z, podczas gdy przy dodaniu dwóch takich samych liczb dodatnich byłby to warunek Nd dodatni.

Rozkazy grup G = 0, 1, 2, 3 – sumowania stałoprzecinkowe

W grupach tych znajdują się rozkazy operacji wykonywanych przez sumator. Jeden argument pochodzi z akumulatora, a drugi podawany jest na wejście x z pamięci, rejestru, lub rozkazu (argument bezpośredni).

Argumenty pochodzące z krótkich rejestrów, B1, B2, B3, B4, B5 i B6, oraz argument bezpośredni N (np. liczba 15 w rozkazie A=A+15;) są liczbami 13-bitowymi bez znaku, przy czym przed wykonaniem operacji uzupełniane są automatycznie zerami do pełnej 39-bitowej długości w ten sposób, że te 13 bitów zajmują pozycje 9..21 słowa. Również wynik posłany do krótkiego rejestru jest obcinany tak, że w rejestrze zapisują się jedynie bity 9..21 słowa wynikowego.

Argumenty w rejestrach: x, s, A, B7 (inaczej M) i R, oraz w pamięci [N] mają pełną 39-bitową długość. Wszystkie długie rejestry i komórki pamięci mają też dodatkowy, czterdziesty, bit techniczny, który z programistycznego punktu widzenia nie ma znaczenia, ale którego wykorzystanie można zauważyć podczas wykonywania operacji mnożenia i dzielenia.

Triada P, prócz tego że wyznacza argument x, decyduje o tym, czy wynik operacji jest posyłany do akumulatora A (P=4..7), czy nie (P=0..3). Niezależnie od tego również triada G wskazuje, dokąd ma być posłany wynik operacji. Łącznie wynik operacji może być posłany do akumulatora i/lub do pamięci, rejestru B, lub rejestru R.

Jeśli wynik nigdzie nie jest posyłany, to jedynym skutkiem wykonania rozkazu jest kod warunku ustawiany wg stanu wyjścia s sumatora: Ujemne, Zerowe, Nadmiar stałoprzecinkowy.

Wynik na wyjściu **s** sumatora		
T	s=	
0	**x**	pobranie
1	**-x**	zmiana znaku
2	**|x|**	wartość bezwzględna
3	**A-x**	różnica
4	**A+x**	suma
5	**x-A**	różnica odwrotna
6	**A&x**	AND – koniunkcja
7	**A^x**	XOR – różnica symetryczna

gdzie **x** zależy od P=				Wynik
0	1	2	3	
0	**[N]**	**B**	**N**	-

4	5	6	7	Wynik
0	**[N]**	**B**	**N**	**A=s**

Zależnie od grupy G wykonuje się ponadto		
G=		
0	-	
1	**[N]= s**	podstawienie
2	**B = s**	podstawienie
3	**R = s**	podstawienie
4	**AM = A*x**	mnożenie
5	**A = A/x**	dzielenie
6	**Rozkazy we/wy, skoków, ...**	
7	**Rozkazy zmiennoprzecinkowe**	

Dla G=0 wynik jest posyłany co najwyżej do akumulatora A, ale ponadto nigdzie indziej.

Dla G=1 wynik jest posyłany ponadto do komórki pamięci.

HAK: W rozkazach TPG=x11 i TPG=x51 wartość wynikowa nie jest posyłana do pamięci, gdyż argument wejściowy pochodzi z pamięci, a czekanie na tę samą komórkę na bębnie w celu zapisania wyniku trwałoby zbyt długo.

Dla G=2 wynik jest posyłany ponadto do rejestru B.

Rejestr B może więc w jednym rozkazie posłużyć do wykonania B-modyfikacji rozkazu, zawierać argument wejściowy operacji, oraz przyjmować wynik operacji.

Dla G=3 wynik jest posyłany ponadto do rejestru rozkazów R do wykonania.

Rozkaz wysyłany do rejestru R nie podlega B-modyfikacji i jest wykonywany począwszy od etapu c) cyklu rozkazowego. Emulator traktuje taki rozkaz jako oddzielny od rozkazu posyłającego.

Rozkazy grupy G = 4 – mnożenia stałoprzecinkowe

Te rozkazy wykonują się dwufazowo:

- najpierw wykonuje się obliczenie tak, jak w grupach 0–3, wynik przesyłany jest do akumulatora albo nie, równolegle argument x wpisywany jest jako mnożnik do rejestru M. Ustawiane są warunki zgodnie ze stanem wyjścia s sumatora.
- następnie wykonuje się mnożenie stałoprzecinkowe zawartości akumulatora A (pozostałej po pierwszej fazie) i mnożnika M, a 77-bitowy iloczyn zapisywany jest w parze rejestrów AM (bit znaku i najstarsze bity wyniku w A, pozostałe bity iloczynu na pozycjach 1..38 rejestru M, bit znaku rejestru M zostaje wyzerowany). Najstarszy bit dla którego zabrakło miejsca w A, tj. bit nr 1 w M, jest wpisywany również do rejestru zaokrągleń Ω. Warunki ustawiane są na nowo zgodnie ze stanem iloczynu w AM.

Jeśli w pierwszej fazie powstanie nadmiar, to druga faza nie jest wykonywana. Jeśli wynik pierwszej fazy nie zostanie umieszczony w A, może on jedynie posłużyć do wykrycia nadmiaru, który uniemożliwi wykonanie mnożenia.

Mnożenie wykonuje się zgodnie z pierwszym algorytmem Robertsona (patrz: Dodatek), w którym czynniki i iloczyn mają kropkę binarną tuż za bitem znaku. Rejestr R służy do przechowywania mnożnej podzielonej przez 2.

Rzeczywista maszyna w jednym kroku rozkazowym wykonywała 2 sumowania. Emulator ukazuje je kolejno w dwóch półkrokach – efekt jest taki, jak gdyby w jednym kroku odbywało się jedno sumowanie, ale krok taki trwa nie 40, a 20 taktów.

Rozkazy grupy G = 5 – dzielenia stałoprzecinkowe

Te rozkazy wykonują się dwufazowo:

- najpierw wykonuje się obliczenie tak, jak w grupach 0–3, wynik przesyłany jest do akumulatora albo nie, równolegle argument x wpisywany jest jako dzielnik do rejestru M. Ustawiane są warunki zgodnie ze stanem wyjścia s sumatora.
- następnie wykonuje się dzielenie stałoprzecinkowe zawartości akumulatora A (pozostałej po pierwszej fazie) i dzielnika M; 40-bitowy iloraz jest zapisywany w A (39 bitów) i w rejestrze zaokrągleń Ω (40. bit), a dzielnik pozostaje niezmieniony w M. Warunki ustawiane są na nowo zgodnie ze stanem ilorazu w A.

Jeśli w pierwszej fazie powstanie nadmiar albo wynik dzielenia byłby nadmiarowy, to druga faza nie jest wykonywana. Jeśli wynik pierwszej fazy nie zostanie umieszczony w A, może on jedynie posłużyć do wykrycia nadmiaru, który uniemożliwi wykonanie dzielenia.

Dzielenie wykonuje się zgodnie z tzw. nierestytucyjną metodą dzielenia stałoprzecinkowego (patrz: Dodatek), w której dzielna i dzielnik mają kropkę binarną tuż za bitem znaku. Rejestr R służy do przechowywania kolejnych reszt częściowych.

Rozkazy grupy G = 6 – przesunięcia, we/wy, skoki

Jest to mniej regularna grupa rozkazów, z których każdemu nadano mnemonik. Zmieniono je tu nieco w stosunku do oryginalnych tak, by jeszcze bardziej kojarzyły się z wykonywaną operacją.

Kolorem niebieskim oznaczono rozkazy przesyłania blokowego dostępne dopiero w ODRZE 1013, a zielonym rozkazy dostępne w ODRZE UMCS, tj. wejścia/wyjścia 8-kanałowego (ich mnemoniki mają na początku znak X) i skok przy braku gotowości, oraz podkreślono, że w miejsce konwertera analogowo-cyfrowego zainstalowano drugi czytnik. Kolorem czerwonym zaznaczono rozkaz niezdefiniowany (o mnemoniku Nic), ale wybrany jako najbardziej odpowiedni do wykonywania skoków bez ustawiania warunków.

Rozkazy grupy G = 7 – operacje zmiennoprzecinkowe

Pierwszym argumentem operacji zmiennoprzecinkowych jest zawartość rejestru AC, czyli pary rejestrów: Am (akumulator mantysy, którym jest rejestr A), oraz Ac (7-bitowy akumulator cechy). Drugi argument pochodzi z pamięci lub B-rejestru. Słowo zmiennoprzecinkowe w pamięci lub w rejestrze B ma cechę zapisaną na ostatnich 7 bitach (bity 32..38), co skraca mantysę do 32 bitów.

Mnożenie i dzielenie jest wykonywane na mantysach tak samo jak w operacjach stałoprzecinkowych. Podczas dzielenia mantys może wystąpić dzielenie przez zero, które powoduje zatrzymanie maszyny. Podczas dzielenia i mnożenia, jak również przy dodawaniach i odejmowaniach może wystąpić nadmiar zmiennoprzecinkowy, który również powoduje zatrzymanie maszyny; drugie wskazanie nadmiaru w opisie rozkazu odnosi się również do nadmiaru cechy.

Dwa najmłodsze bity triady P wskazują, czy na zakończenie operacji ma być wykonana normalizacja i zaokrąglenie logiczne liczby zmiennoprzecinkowej:

`P=.00 = BON`	–	bez normalizacji	i	bez zaokrąglenia
`P=.01 = BO`	–	z normalizacją	ale	bez zaokrąglenia
`P=.10 = BN`	–	bez normalizacji	ale	z zaokrągleniem
`P=.11 = –`	–	z normalizacją	i	z zaokrągleniem

Zaokrąglenie logiczne polega na umieszczeniu na bicie 31 sumy logicznej (alternatywy) bitów 31 i 32. Jeśli bit 31 jest jedynką lub bit 32 jest zerem, to zaokrąglenie logiczne niczego nie zmienia.

W operacjach pobierania zmiennoprzecinkowego, dla TPG=`1x7`, zaokrąglenie logiczne nie pociąga żadnych skutków.

HAK: W operacjach posyłania zmiennoprzecinkowego, tj. dla TPG=`2x7`, nie wykonuje się zaokrąglenia i normalizacji.

Tabelaryczny wykaz rozkazów

G=0	P 0 4	P 1 5	P 2 6	P 3 7
T 0	0; ‖0Z0 A=0; ‖0Z0	[N]; ‖uz0 A=[N]; ‖uz0	B; ‖uz0 A=B; ‖uz0	N; ‖0z0 A=N; ‖0z0
T 1	-0; ‖0Z0 A=-0; ‖0Z0	-[N]; ‖uzv A=-[N]; ‖uzv	-B; ‖uzv A=-B; ‖uzv	-N; ‖uz0 A=-N; ‖uz0
T 2	|0|; ‖0Z0 A=|0|; ‖0Z0	|[N]|; ‖0zv A=|[N]|; ‖0zv	|B|; ‖0zv A=|B|; ‖0zv	|N|; ‖0z0 A=|N|; ‖0z0
T 3	A-0; ‖uz0 A=A-0; ‖uz0	A-[N]; ‖uzv A=A-[N]; ‖uzv	A-B; ‖uzv A=A-B; ‖uzv	A-N; ‖uzv A=A-N; ‖uzv
T 4	A+0; ‖uz0 A=A+0; ‖uz0	A+[N]; ‖uzv A=A+[N]; ‖uzv	A+B; ‖uzv A=A+B; ‖uzv	A+N; ‖uzv A=A+N; ‖uzv
T 5	0-A; ‖uzv A=0-A; ‖uzv	[N]-A; ‖uzv A=[N]-A; ‖uzv	B-A; ‖uzv A=B-A; ‖uzv	N-A; ‖uzv A=N-A; ‖uzv
T 6	A&0; ‖0Z0 A=A&0; ‖0Z0	A&[N]; ‖uz0 A=A&[N]; ‖uz0	A&B; ‖uz0 A=A&B; ‖uz0	A&N; ‖uz0 A=A&N; ‖uz0
T 7	A^0; ‖uz0 A=A^0; ‖uz0	A^[N]; ‖uz0 A=A^[N]; ‖uz0	A^B; ‖uz0 A=A^B; ‖uz0	A^N; ‖uz0 A=A^N; ‖uz0

G=1	P 0 4	P 1 5	P 2 6	P 3 7
T 0	[N] =0; ‖0Z0 [N]=A=0; ‖0Z0	[N]; ‖uz0 HAK A=[N]; ‖uz0 HAK	[N] =B; ‖uz0 [N]=A=B; ‖uz0	[N] =N; ‖0z0 [N]=A=N; ‖0z0
T 1	[N] =-0; ‖0Z0 [N]=A=-0; ‖0Z0	-[N]; ‖uzv HAK A=-[N]; ‖uzv HAK	[N] =-B; ‖uzv [N]=A=-B; ‖uzv	[N] =-N; ‖uz0 [N]=A=-N; ‖uz0
T 2	[N] =|0|; ‖0Z0 [N]=A=|0|; ‖0Z0	|[N]|; ‖0zv HAK A=|[N]|; ‖0zv HAK	[N] =|B|; ‖0zv [N]=A=|B|; ‖0zv	[N] =|N|; ‖0z0 [N]=A=|N|; ‖0z0
T 3	[N] =A-0; ‖uz0 [N]=A=A-0; ‖uz0	A-[N]; ‖uzv HAK A=A-[N]; ‖uzv HAK	[N] =A-B; ‖uzv [N]=A=A-B; ‖uzv	[N] =A-N; ‖uzv [N]=A=A-N; ‖uzv
T 4	[N] =A+0; ‖uz0 [N]=A=A+0; ‖uz0	A+[N]; ‖uzv HAK A=A+[N]; ‖uzv HAK	[N] =A+B; ‖uzv [N]=A=A+B; ‖uzv	[N] =A+N; ‖uzv [N]=A=A+N; ‖uzv
T 5	[N] =0-A; ‖uzv [N]=A=0-A; ‖uzv	[N]-A; ‖uzv HAK A=[N]-A; ‖uzv HAK	[N] =B-A; ‖uzv [N]=A=B-A; ‖uzv	[N] =N-A; ‖uzv [N]=A=N-A; ‖uzv
T 6	[N] =A&0; ‖0Z0 [N]=A=A&0; ‖0Z0	A&[N]; ‖uz0 HAK A=A&[N]; ‖uz0 HAK	[N] =A&B; ‖uz0 [N]=A=A&B; ‖uz0	[N] =A&N; ‖uz0 [N]=A=A&N; ‖uz0
T 7	[N] =A^0; ‖uz0 [N]=A=A^0; ‖uz0	A^[N]; ‖uz0 HAK A=A^[N]; ‖uz0 HAK	[N] =A^B; ‖uz0 [N]=A=A^B; ‖uz0	[N] =A^N; ‖uz0 [N]=A=A^N; ‖uz0

HAK: W rozkazach TPG=x11 i TPG=x51 wartość wynikowa nie jest posyłana do pamięci, gdyż argument wejściowy pochodzi z pamięci, a czekanie na tę samą komórkę na bębnie w celu zapisania wyniku trwałoby zbyt długo

G=2	P 0 4	1 5	2 6	3 7
T 0	B= 0; ‖0Z0 B=A=0; ‖0Z0	B= [N]; ‖uz0 B=A=[N]; ‖uz0	B= B; ‖uz0 B=A=B; ‖uz0	B= N; ‖0z0 B=A=N; ‖0z0
1	B= -0; ‖0Z0 B=A=-0; ‖0Z0	B= -[N]; ‖uzv B=A=-[N]; ‖uzv	B= -B; ‖uzv B=A=-B; ‖uzv	B= -N; ‖uz0 B=A=-N; ‖uz0
2	B= |0|; ‖0Z0 B=A=|0|; ‖0Z0	B= |[N]|; ‖0zv B=A=|[N]|; ‖0zv	B= |B|; ‖0zv B=A=|B|; ‖0zv	B= |N|; ‖0z0 B=A=|N|; ‖0z0
3	B= A-0; ‖uz0 B=A=A-0; ‖uz0	B= A-[N]; ‖uzv B=A=A-[N]; ‖uzv	B= A-B; ‖uzv B=A=A-B; ‖uzv	B= A-N; ‖uzv B=A=A-N; ‖uzv
4	B= A+0; ‖uz0 B=A=A+0; ‖uz0	B= A+[N]; ‖uzv B=A=A+[N]; ‖uzv	B= A+B; ‖uzv B=A=A+B; ‖uzv	B= A+N; ‖uzv B=A=A+N; ‖uzv
5	B= 0-A; ‖uzv B=A=0-A; ‖uzv	B= [N]-A; ‖uzv B=A=[N]-A; ‖uzv	B= B-A; ‖uzv B=A=B-A; ‖uzv	B= N-A; ‖uzv B=A=N-A; ‖uzv
6	B= A&0; ‖0Z0 B=A=A&0; ‖0Z0	B= A&[N]; ‖uz0 B=A=A&[N]; ‖uz0	B= A&B; ‖uz0 B=A=A&B; ‖uz0	B= A&N; ‖uz0 B=A=A&N; ‖uz0
7	B= A^0; ‖uz0 B=A=A^0; ‖uz0	B= A^[N]; ‖uz0 B=A=A^[N]; ‖uz0	B= A^B; ‖uz0 B=A=A^B; ‖uz0	B= A^N; ‖uz0 B=A=A^N; ‖uz0

G=3	P 0 4	1 5	2 6	3 7
T 0	R= 0; ‖0Z0 R=A=0; ‖0Z0	R= [N]; ‖uz0 R=A=[N]; ‖uz0	R= B; ‖uz0 R=A=B; ‖uz0	R= N; ‖0z0 R=A=N; ‖0z0
1	R= -0; ‖0Z0 R=A=-0; ‖0Z0	R= -[N]; ‖uzv R=A=-[N]; ‖uzv	R= -B; ‖uzv R=A=-B; ‖uzv	R= -N; ‖uz0 R=A=-N; ‖uz0
2	R= |0|; ‖0Z0 R=A=|0|; ‖0Z0	R= |[N]|; ‖0zv R=A=|[N]|; ‖0zv	R= |B|; ‖0zv R=A=|B|; ‖0zv	R= |N|; ‖0z0 R=A=|N|; ‖0z0
3	R= A-0; ‖uz0 R=A=A-0; ‖uz0	R= A-[N]; ‖uzv R=A=A-[N]; ‖uzv	R= A-B; ‖uzv R=A=A-B; ‖uzv	R= A-N; ‖uzv R=A=A-N; ‖uzv
4	R= A+0; ‖uz0 R=A=A+0; ‖uz0	R= A+[N]; ‖uzv R=A=A+[N]; ‖uzv	R= A+B; ‖uzv R=A=A+B; ‖uzv	R= A+N; ‖uzv R=A=A+N; ‖uzv
5	R= 0-A; ‖uzv R=A=0-A; ‖uzv	R= [N]-A; ‖uzv R=A=[N]-A; ‖uzv	R= B-A; ‖uzv R=A=B-A; ‖uzv	R= N-A; ‖uzv R=A=N-A; ‖uzv
6	R= A&0; ‖0Z0 R=A=A&0; ‖0Z0	R= A&[N]; ‖uz0 R=A=A&[N]; ‖uz0	R= A&B; ‖uz0 R=A=A&B; ‖uz0	R= A&N; ‖uz0 R=A=A&N; ‖uz0
7	R= A^0; ‖uz0 R=A=A^0; ‖uz0	R= A^[N]; ‖uz0 R=A=A^[N]; ‖uz0	R= A^B; ‖uz0 R=A=A^B; ‖uz0	R= A^N; ‖uz0 R=A=A^N; ‖uz0

G=4	P			
	0 4	1 5	2 6	3 7
T 0	M=0, M, AM=A*M; Ω‖0Z00 M=0, A=M, AM=A*M; Ω‖0Z00	M=[N], M, AM=A*M; Ω‖uz0n M=[N], A=M, AM=A*M; Ω‖uz0n	M=B, M, AM=A*M; Ω‖uz0n M=B, A=M, AM=A*M; Ω‖uz0n	M=N, M, AM=A*M; Ω‖uz00 M=N, A=M, AM=A*M; Ω‖0z00
1	M=0, -M, AM=A*M; Ω‖0Z00 M=0, A=-M, AM=A*M; Ω‖0Z00	M=[N], -M, AM=A*M; Ω‖uzvn M=[N], A=-M, AM=A*M; Ω‖uzvn	M=B, -M, AM=A*M; Ω‖uzvn M=B, A=-M, AM=A*M; Ω‖uzvn	M=N, -M, AM=A*M; Ω‖uz00 M=N, A=-M, AM=A*M; Ω‖uz00
2	M=0, |M|, AM=A*M; Ω‖0Z00 M=0, A=|M|, AM=A*M; Ω‖0Z00	M=[N], |M|, AM=A*M; Ω‖uzvn M=[N], A=|M|, AM=A*M; Ω‖uzvn	M=B, |M|, AM=A*M; Ω‖uzvn M=B, A=|M|, AM=A*M; Ω‖uzvn	M=N, |M|, AM=A*M; Ω‖uz00 M=N, A=|M|, AM=A*M; Ω‖uz00
3	M=0, A-M, AM=A*M; Ω‖0Z00 M=0, A=A-M, AM=A*M; Ω‖0Z00	M=[N], A-M, AM=A*M; Ω‖uzvn M=[N], A=A-M, AM=A*M; Ω‖uzvn	M=B, A-M, AM=A*M; Ω‖uzvn M=B, A=A-M, AM=A*M; Ω‖uzvn	M=N, A-M, AM=A*M; Ω‖uzv0 M=N, A=A-M, AM=A*M; Ω‖uzv0
4	M=0, A+M, AM=A*M; Ω‖0Z00 M=0, A=A+M, AM=A*M; Ω‖0Z00	M=[N], A+M, AM=A*M; Ω‖uzvn M=[N], A=A+M, AM=A*M; Ω‖uzvn	M=B, A+M, AM=A*M; Ω‖uzvn M=B, A=A+M, AM=A*M; Ω‖uzvn	M=N, A+M, AM=A*M; Ω‖uzv0 M=N, A=A+M, AM=A*M; Ω‖uzv0
5	M=0, M-A, AM=A*M; Ω‖uzv0 M=0, A=M-A, AM=A*M; Ω‖uzv0	M=[N], M-A, AM=A*M; Ω‖uzvn M=[N], A=M-A, AM=A*M; Ω‖uzvn	M=B, M-A, AM=A*M; Ω‖uzvn M=B, A=M-A, AM=A*M; Ω‖uzvn	M=N, M-A, AM=A*M; Ω‖uzv0 M=N, A=M-A, AM=A*M; Ω‖uzv0
6	M=0, A&M, AM=A*M; Ω‖0Z00 M=0, A=A&M, AM=A*M; Ω‖0Z00	M=[N], A&M, AM=A*M; Ω‖uz0n M=[N], A=A&M, AM=A*M; Ω‖uz0n	M=B, A&M, AM=A*M; Ω‖uz0n M=B, A=A&M, AM=A*M; Ω‖uz0n	M=N, A&M, AM=A*M; Ω‖uz00 M=N, A=A&M, AM=A*M; Ω‖uz00
7	M=0, A^M, AM=A*M; Ω‖0Z00 M=0, A=A^M, AM=A*M; Ω‖0Z00	M=[N], A^M, AM=A*M; Ω‖uz0n M=[N], A=A^M, AM=A*M; Ω‖uz0n	M=B, A^M, AM=A*M; Ω‖uz0n M=B, A=A^M, AM=A*M; Ω‖uz0n	M=N, A^M, AM=A*M; Ω‖uz00 M=N, A=A^M, AM=A*M; Ω‖uz00

G=5	P			
	0 4	1 5	2 6	3 7
T 0	M=0, M, A=A/M; Ω‖0Z0● M=0, A=M, A=A/M; Ω‖0Z0●	M=[N], M, A=A/M; Ω‖uz0/ M=[N], A=M, A=A/M; Ω‖uz0/	M=B, M, A=A/M; Ω‖uz0/ M=B, A=M, A=A/M; Ω‖uz0/	M=N, M, A=A/M; Ω‖uz0/ M=N, A=M, A=A/M; Ω‖0z0N
1	M=0, -M, A=A/M; Ω‖0Z0● M=0, A=-M, A=A/M; Ω‖0Z0●	M=[N], -M, A=A/M; Ω‖uzv/ M=[N], A=-M, A=A/M; Ω‖uzv/	M=B, -M, A=A/M; Ω‖uzv/ M=B, A=-M, A=A/M; Ω‖uzv/	M=N, -M, A=A/M; Ω‖uz0/ M=N, A=-M, A=A/M; Ω‖uz0/
2	M=0, |M|, A=A/M; Ω‖0Z0● M=0, A=|M|, A=A/M; Ω‖0Z0●	M=[N], |M|, A=A/M; Ω‖uzv/ M=[N], A=|M|, A=A/M; Ω‖uzv/	M=B, |M|, A=A/M; Ω‖uzv/ M=B, A=|M|, A=A/M; Ω‖uzv/	M=N, |M|, A=A/M; Ω‖uz0/ M=N, A=|M|, A=A/M; Ω‖uz0N
3	M=0, A-M, A=A/M; Ω‖uz0● M=0, A=A-M, A=A/M; Ω‖uz0●	M=[N], A-M, A=A/M; Ω‖uzv/ M=[N], A=A-M, A=A/M; Ω‖uzv/	M=B, A-M, A=A/M; Ω‖uzv/ M=B, A=A-M, A=A/M; Ω‖uzv/	M=N, A-M, A=A/M; Ω‖uzv/ M=N, A=A-M, A=A/M; Ω‖uzv/
4	M=0, A+M, A=A/M; Ω‖uz0● M=0, A=A+M, A=A/M; Ω‖uz0●	M=[N], A+M, A=A/M; Ω‖uzv/ M=[N], A=A+M, A=A/M; Ω‖uzv/	M=B, A+M, A=A/M; Ω‖uzv/ M=B, A=A+M, A=A/M; Ω‖uzv/	M=N, A+M, A=A/M; Ω‖uzv/ M=N, A=A+M, A=A/M; Ω‖uzv/
5	M=0, M-A, A=A/M; Ω‖uzv● M=0, A=M-A, A=A/M; Ω‖uzv●	M=[N], M-A, A=A/M; Ω‖uzv/ M=[N], A=M-A, A=A/M; Ω‖uzv/	M=B, M-A, A=A/M; Ω‖uzv/ M=B, A=M-A, A=A/M; Ω‖uzv/	M=N, M-A, A=A/M; Ω‖uzv/ M=N, A=M-A, A=A/M; Ω‖uzv/
6	M=0, A&M, A=A/M; Ω‖0Z0● M=0, A=A&M, A=A/M; Ω‖0Z0●	M=[N], A&M, A=A/M; Ω‖uz0/ M=[N], A=A&M, A=A/M; Ω‖uz0/	M=B, A&M, A=A/M; Ω‖uz0/ M=B, A=A&M, A=A/M; Ω‖uz0/	M=N, A&M, A=A/M; Ω‖uz0/ M=N, A=A&M, A=A/M; Ω‖uz0/
7	M=0, A^M, A=A/M; Ω‖uz0● M=0, A=A^M, A=A/M; Ω‖uz0●	M=[N], A^M, A=A/M; Ω‖uz0/ M=[N], A=A^M, A=A/M; Ω‖uz0/	M=B, A^M, A=A/M; Ω‖uz0/ M=B, A=A^M, A=A/M; Ω‖uz0/	M=N, A^M, A=A/M; Ω‖uz0/ M=N, A=A^M, A=A/M; Ω‖uz0/

W rozkazach mnożenia i dzielenia argument wchodzący na wejście x sumatora posyłany jest jednocześnie do rejestru M. Wykorzystano ten fakt w zapisie rozkazów – w operacjach sumowania wykonywanych przed mnożeniem lub dzieleniem wskazano rejestr M zamiast tego argumentu. Dzięki temu zabiegowi uzyskano bardziej zwarty i przejrzysty zapis funkcji rozkazów, jak również uniknięto błędnej sugestii, jakoby argument pamięciowy był pobierany z pamięci dwukrotnie.

G=6	P							
T	0	1	2	3	4	5	6	7
0	–	**LL N;** A=A<<<N; ‖uz0 Przesunięcie logiczne w lewo	**We0 N;** **We0;** A_{34-38}=PTR0; ‖uz. Czytanie kwintetu z czytnika pierwszego	–	**SkZ N;** if (0Zv) goto N; Skok przy zerze	–	**XWe0 N;** **XWe0;** A_{31-38}=PTR0'; ‖uz. Czytanie oktetu z czytnika pierwszego	–
1	–	**LP N;** A=A>>>N; Ω‖uz0 Przesunięcie logiczne w prawo	**We1 N;** **We1;** A_{34-38}=TTY; ‖uz. Czytanie kwintetu z dalekopisu	**BF61 B,N;** Ferryt61<--N(1+B); Przesyłanie (B+1) komórek na pierwszą ścieżkę ferrytową	**SkU N;** if (U0v) goto N; Skok przy ujemnym	–	–	–
2	–	**AL N;** A=A<<N; ‖uzv Przesunięcie arytmetyczne w lewo	**We2 N;** **We2;** A_{34-38}=PTR2; ‖uz. Czytanie kwintetu z czytnika drugiego	**BF62 B,N;** Ferryt62<--N(1+B); Przesyłanie (B+1) komórek na drugą ścieżkę ferrytową	**SkD N;** if (00v) goto N; Skok przy dodatnim	–	**XWe2 N;** **XWe2;** A_{31-38}=PTR2'; ‖uz. Czytanie oktetu z czytnika drugiego	–
3	–	**AP N;** A=A>>N; Ω‖uz0 Przesunięcie arytmetyczne w prawo	**CzK N;** **CzK;** A=przyciskiA; ‖uz. Czytanie z pulpitu maszyny stanu przycisków akumulatora	**FB61 B,N;** Ferryt61-->N(1+B); Przesyłanie (B+1) komórek z pierwszej ścieżki ferrytowej	**SkV N;** if (uzV) goto N; Skok przy nadmiarze	–	–	–
4	–	**CP N;** A=A>><N; Ω‖uz0 Przesunięcie cykliczne w prawo	**Okr N;** **Okr;** A=A+Ω; ‖uzv Zaokrąglenie normalne (dodanie bitu Ω do A)	**FB62 B,N;** Ferryt62-->N(1+B); Przesyłanie (B+1) komórek z drugiej ścieżki ferrytowej	**SkBG B,N;** if (BGWeB) goto N; Skok przy braku gotowości wejścia 5-kanałowego nr B	–	–	–
5	–	**ALD N;** AM=AM<<N; Ω‖uzv Przesunięcie długie arytmetyczne w lewo	**Wy5 N;** **Wy5;** PTP5=A_{0-4}; ‖uz. Pisanie kwintetu na perforator	–	**SkLC B--,N;** if (B--) goto N; Skok z licznikiem cykli -- gdy było B ≠ 0 to skok do N	–	**XWy5 N;** **XWy5;** PTP5'=A_{0-7}; ‖uz. Pisanie oktetu na perforator / samopis	–
6	–	**APD N;** AM=AM>>N; Ω‖uz0 Przesunięcie długie arytmetyczne w prawo	**Wy6 N;** **Wy6;** TTY=A_{0-4}; ‖uz. Pisanie kwintetu na dalekopis	–	**SkLC B++,N;** if (B++) goto N; Skok z licznikiem cykli ++ gdy było B ≠ 0 to skok do N	–	–	–
7	–	**DzD N;** A=A//M,N; Ω‖uzv/ Dzielenie długie stałoprzecinkowe (N-3 bity ilorazu)	**Stop N;** **Stop;** Et: Stop, goto K; Zatrzymanie maszyny: K ≠ Et – niestabilne K = Et – stabilne	–	**SkS N;** [N] = :000 N K B0, goto N+2; Skok ze śladem: rozkaz powrotu do [N], i skok pod adres N+2	–	**Nic N;** **Nic;** Nic Nie Rób, goto K; Rozkaz predestynowany jako Skok bez ustawiania warunków	–

W rozkazach T26 i T66 pokazano opcjonalny, obojętny argument N. W innych rozkazach, w których argument N jest obojętny, nie jest ukazywany z uwagi na przyjęty sposób zapisu.

Taki zbędny argument N może być przydatny w rozkazie STOP, oraz w rozkazach wejścia, gdy odpowiednio dobrana wartość N, wyświetlana w części AR rejestru rozkazów, sygnalizuje operatorowi stan programu.

W rozkazie SkBG, argument B nie wyznacza rejestru, lecz numer urządzenia wejściowego, tym samym musi być zgodny z ew. rejestrem B modyfikującym rozkaz.

G=7		P							
		0 **BON**	1 **BO** *tylko Norm.*	2 **BN** *tylko Zaokr.*	3 **–** *Norm.+Zaokr.*	4 **BON**	5 **BO** *tylko Norm.*	6 **BN** *tylko Zaokr.*	7 **–** *Norm.+Zaokr.*
T	0	–				–			
	1	AC = [N];		‖uz.0		AC = B;		‖uz.0	
	2	[N] = AC;		‖uz.0 HAK		B = AC;		‖uz.0 HAK	
	3	AC = AC + [N];		‖uz.*		AC = AC + B;		‖uz.*	
	4	AC = AC - [N];		‖uz.*		AC = AC - B;		‖uz.*	
	5	AC = [N] - AC;		‖uz.*		AC = B - AC;		‖uz.*	
	6	AC = AC * [N];		M=#‖uz.*		AC = AC * B;		M=#‖uz.*	
	7	AC = AC / [N];		M=#‖uz/*		AC = AC / B;		M=#‖uz/*	

HAK: W operacjach posyłania zmiennoprzecinkowego, tj. dla TPG=2x7, nie wykonuje się zaokrąglenia i normalizacji.

Oprogramowanie

Aktualnie nie mam żadnego oryginalnego programu na ODRĘ. Wykonane przeze mnie rekonstrukcje programów, np. programu STAŁEGO, oraz drobne programy demonstracyjne znajdują się wraz z dokumentacją w programotece.

Inne cenne programy, jak np.:

- język programowania PJZ – Podstawowy Język Zewnętrzny. Czy to jest to samo co PODSTAWOWY 03 – 03-IV-1,2 (WR-Z-1,2), lub 03-IV-1,2 WYD. SP 1 ST, albo PODSTAWOWY 03 P – 03-IV-83,84 (WR-Z-10,11), albo PODSTAWOWY 013 – 013-IV-1,2, lub 013-IV-1,2 WYD. SP 1 ST (WR-Z-20,21),
- język programowania JAS – Język Adresów Symbolicznych,
- autokod MOST I (03-VI-2) – język programowania,
- autokod MOST F (013-VI-1) – język programowania,
- program BILL (WR-Z-30) – szybkie wczytywanie taśm w kodzie PIĄTKOWYM,
- program DECODER (WR-Z-7) – dekodowanie taśm w kodzie PIĄTKOWYM,
- program FORT (WR-OCT-1) – wczytywanie taśm w kodzie ósemkowym THETA,
- program ANTYFORT (WR-OCT-2) – perforowanie taśm w kodzie ósemkowym THETA,
- symulator TYMAC – symulator typowej maszyny cyfrowej,
- system operacyjny PROM 03 (WR-Z-25 : 03-IV-141) do testowania programów,
- system operacyjny PROM 013 (WR-Z-26 : 013-IV-13) do testowania programów,
- język programowania FALA-69 (WR-TR-7) – opracowany w ZMN UMCS,
- program FALCONET – szybki translator „próbny” programów w języku FALA,
- system MIX (WR-Z-44) – system operacyjny do testowania programów w języku FALA,
- interpreter działań na macierzach 03-IV-39 (WR-TR-5),
- program KORD (WR-Z-46) szybkiego poprawiania taśm z danymi do innych programów,
- program KORAL (WR-Z-49) poprawiania taśm zawierających też teksty alfanumeryczne,
- podprogramy obliczania wartości funkcji jednej zmiennej,
- programy i podprogramy algebraiczne,
- programy z zakresu statystyki matematycznej,
- pakiet podprogramów arytmetyki podwójnej precyzji,
- programy z zakresu programowania liniowego, itp.,

a w szczególności większe z nich, prawdopodobnie nie zostaną odtworzone.

Oryginalne programy miały oznaczenia według różnej systematyki opisanej w [2]. W ich rekonstrukcjach zamieszczam te oznaczenia. Oto przykładowe oznaczenia programu STAŁEGO zapisanego na bębnie:

wg systematyki Elwro:

(03-IV-35)	03=Odra1003, IV=PODSTAWOWY, 35=numer ewidencyjny	013=Odra1013, VI=MOST-1

wg systematyki ZMN UMCS:

WR-Z-5	W=PODSTAWOWY, R=program Z=programy we/wy 5=numer ewidencyjny	X=MOST-1 S=podprogram	F=FALA-69

Uwagi, wątpliwości i rozstrzygnięcia

Na potrzeby emulatora skrypt akademicki UMCS jest w zasadzie wyczerpujący, chociaż zdarzają się wątpliwości zrozumiałe przy szczegółowym odwzorowywaniu pracy maszyny. Należy zwrócić uwagę na zastrzeżenie autora, iż na potrzeby dydaktyczne opis maszyny został nieco uproszczony: pominięto istnienie niektórych rejestrów i układów wewnętrznych niedostępnych dla programisty.

1) **Dodatkowy bęben pamięci zewnętrznej**
   Nie zaimplementowano zapowiadanego w [2] dodatkowego bębna pamięci zewnętrznej – brak opisu.

2) **Pamięć operacyjna**
   Pamięć na bębnie magnetycznym zachowuje zawartość aż (albo tylko) do zamknięcia lub odświeżenia strony z emulatorem. Wyłączenie maszyny czy zasilania nie kasuje jej. Pamięć ferrytowa zachowuje zawartość tylko do wyłączenia maszyny.

3) **Pamięć STAŁA**
   Program STAŁY jest rekonstrukcją, zatem zdecydowałem, by dało się go łatwo, różnymi sposobami, wymienić na lepszy. Przełączenie maszyny na model ODRA 1003 zdejmuje ochronę pamięci programu STAŁEGO przed zapisem (pozwala wgrać inny program STAŁY). Ponieważ jest to ścieżka bębnowa, to można po wgraniu wyłączyć maszynę (a nawet zasilanie) i zmienić model na wyższy, co znowu założy ochronę pamięci stałej.

4) **Przesyłanie blokowe**
   Nie jest całkowicie jasne, czy i jak operacje przesyłania blokowego działają w reżimie adresacji przeplotowej. Prostota realizacji sprzętowej przemawia za działaniem jedynie w adresacji normalnej, ale nie ma na ten temat żadnej uwagi.
   Emulator przesyła blokowo zawartość komórek również w adresacji przeplotowej, oczywiście tych i tylko tych, których część strefowa adresu mieści się w przedziale wyznaczonym przez rozkaz. W reżimie adresacji normalnej jest to spójny blok komórek, zaś w przeplotowej kopiowany blok komórek jest poprzerywany komórkami nie podlegającymi kopiowaniu.

5) **Przyciski [StartB], [Z], [StopB] i [StopM]**
   Niezależnie od interpretacji informacji w [2], przyciski emulatora [StartB] i [Z] są stabilne, a [StopB] i [StopM] niestabilne i wyciskają te pierwsze. Przyciski [Z] i [StartB] wyskakują na końcu cyklu; podobnie przycisk [StopCPU].

6) **Przycisk [PP]**
   Przycisk [PP] ustawia bęben tuż przed nieprzeplotową strefą nr 127 (dając natychmiastowy dostęp do adresu, a następnie treści pierwszej komórki ścieżki (komórki w strefie 0). Tak jak w rzeczywistej maszynie, wciśnięcie przycisku [PP] w czasie działania programu może zdezorganizować pracę komputera.

7) **Przycisk [D]**
   Dalekopis emulatora nie umożliwia pracy offline, bez komputera; prace wykonywane offline można wykonywać z pomocą prostych programów na ODRĘ. Ponieważ dalekopis nigdy nie zakłóca pracy komputera, to przycisk „Blokada Dalekopisu”, jako zasadniczo zbędny, zaadaptowano do „Detekcji niezgodności Komputer-Program” – czytaj niżej.

8) **Lampka KP i przycisk [D]**
   Zapalenie się lampki KP oznaczało w ODRZE 1013, że maszynę zatrzymał błąd wykryty przez układ kontroli parzystości słowa pobieranego z pamięci ferrytowej.

Emulator nie przewiduje wykrywania błędów charakterystycznych dla ówczesnego sprzętu, za to wychwytuje rozkazy NIC NIE RÓB zdefiniowane inaczej w wyższych modelach ODRY, ze względu na przypuszczenie, iż dany program wymaga innego modelu. Nazwałem tę cechę „Kontrolą Komputer-Program". Odpowiednie objaśnienie w okienku podglądu trzewi CPU sygnalizuje wykonywanie się takiego rozkazu, ale nie zatrzymuje maszyny. Zapalenie się lampki KP i zatrzymanie następuje tylko przy wciśniętym przycisku [D] – „Detekcji niezgodności Komputer-Program".

9) **Lampki NZ i KP**
Lampki NZ i KP, zapalone w wyniku Nadmiaru Zmiennoprzecinkowego lub Kontroli Komputer-Program, gasną dopiero po uruchomieniu maszyny przyciskiem [StartCPU]. Przyciski [zA], [O] i [doA] ich nie gaszą.

10) **Przycisk [StartCPU]**
Przycisk [StartCPU] wciśnięty w czasie pracy CPU nie spowalnia pamięci ferrytowej, który to efekt odnotowano w [2].

11) **Rozkaz z sumatora**
Wynik operacji wysłany do rejestru rozkazów R uważa się za oddzielny rozkaz, a nie kontynuację rozkazu wysyłającego, ze wszystkimi konsekwencjami, jak zakończenie bieżącego cyklu rozkazowego, zliczanie kroków, itp.

12) **Rejestry x i s sumatora**
Prawdopodobnie nie każdy udział rejestrów x (drugi argument) i s (wynik) sumatora jest ukazywany podczas wykonywania rozkazów. Np. emulator nie angażuje rejestrów x i s sumatora w fazę dzielenia stałoprzecinkowego.

13) **Rejestr zaokrągleń Ω**
Zaokrąglenie normalne, rozkazem :426, powoduje dodanie bitu Omega do akumulatora bez angażowania sumatora.
Algorytm dzielenia ustawia rejestr zaokrągleń po każdym kroku; jego wartość pojawia się też na bicie technicznym akumulatora, skąd jest zdejmowana dopiero na końcu rozkazu.
Przy przesunięciu długim w lewo o 0 pozycji bitu Ω nie ustawia się (zgodnie z opisem w [2]), chociaż chciałoby się mieć możliwość kopiowania bitu nr 1 rejestru M do rejestru zaokrągleń!

14) **Przycisk [ŁadR]**
Przycisk [ŁadR] ładuje drugą część adresową NR (K) rejestru rozkazów dopiero po powtórnym wciśnięciu, który to efekt zachodził czasami, jak odnotowano w [2].

15) **Przyciski [ŁadA] i [ZerA]**
Przyciski [ŁadA] i [ZerA] nie ustawiają warunków U, Z i Nd, jak czynią to rozkazy ładowania do akumulatora. To tak gwoli zwrócenia uwagi.

16) **Przyciski [StopAR] i [StopNR]**
Zatrzymanie programu wskutek selektywnego stopu żądanego przyciskami [StopAR] i/lub [StopNR] następuje zawsze, gdy rozkaz ma zgodną skuteczną część adresową niezależnie od jej znaczenia i użycia, nawet gdy jest obojętna w rozkazie.

17) **Bit techniczny**
Bit techniczny rejestrów długich: R (gdy pełni rolę arytmetyczną), A, M, x i s jest ustawiany tylko w trakcie mnożenia, dzielenia, przesunięć długich i dzielenia długiego, jak też w operacjach zmiennoprzecinkowych.

18) **Przesunięcia długie**
Opis rozkazów przesunięć długich jest różny w pracach [2] i [3]. W [2] jest wyraźnie mowa o 6-bitowym liczniku przesunięć długich i 8-bitowym liczniku dzielenia długiego, zaś w [3] określa się maksymalne przesunięcie długie na 225 bitów, co zdaje się być pomyłką edytorską

(chociaż powtórzoną kilkukrotnie) i zapewne chodziło o 255 pozycji bitowych.
Emulator stosuje 8-bitowy licznik: raz, że działanie przesunięć długich jest analogiczne do dzielenia długiego, gdyż przesunięcie długie o 1 bit zajmuje cały krok cyklu procesora, a więc nie ma do niego zastosowania wspomniany w [2] licznik chwil, i dwa, że nawet jeśli istotnie licznik przesunięć długich był 6-bitowy, to jest nadzieja że w odpowiednich rozkazach poprawnych programów nie ustawiano starszych bitów liczby N wyznaczającej wielkość przesunięcia (chociaż nie można wykluczyć stosowania różnych chwytów programistycznych).

19) **Przesunięcia arytmetyczne w lewo**
Zarówno w [2] jak i [3] opis przesunięć arytmetycznych w lewo jest niejasny. Z jednej strony mówi się, że bit znaku nie uczestniczy w przesunięciu, a z drugiej porównuje się przesunięcie z mnożeniem przez potęgę liczby 2. Ponadto jest niezrozumiałe sygnalizowanie nadmiaru przy przesunięciu logicznym tylko na podstawie bitów 0 i 1. Nie podaje się przy tym, jaka wartość powstanie w razie wystąpienia nadmiaru, czy przesunięcie będzie wtedy wykonane. Niejasności te spowodowały przyjęcie w emulatorze powszechnie znanego działania tych operacji: przesunięcia logiczne i arytmetyczne w lewo są zawsze wykonywane i dają identyczne wartości wynikowe, ale logiczne nie wykrywa nadmiaru (zeruje warunek Nd), zaś arytmetyczne wykrywa nadmiar (i ustawia warunek Nd gdy jakikolwiek bit różny od bitu znaku wysunął się na pozycję bitu znaku lub przed rejestr).

20) **Skok ze Śladem**
Opis rozkazu SkS jest różny w pracach [2] i [3]. Wg [3] bit MR (Z) w śladzie jest zerowany, zaś wg [2] pozostaje skopiowany z rozkazu SkS. Emulator jest zgodny z [3], uznając opis w [2] za oczywistą pomyłkę.
Pozostawiany ślad zawiera wartości N, K, B po ewentualnej B-modyfikacji.

21) **Skoki z Licznikiem Cykli**
Rozkazy KC- i KC+ wybierają adres następnego rozkazu na podstawie wartości rejestru B (zerowa/niezerowa) sprzed jego dekrementacji lub inkrementacji. W przypadku rejestru B7 zaznacza się, że dekrementacja/inkrementacja odbywa się jedynie na bitach 9..21 (tj. na części AR) słowa zapisanego w B7, ale brak wyraźnego wskazania, czy tylko te, czy wszystkie bity rejestru B7 decydują, który adres zostanie wybrany jako adres następnego rozkazu. Ponieważ rozkazy KC- i KC+ nie ustawiają warunków U, Z i Nd, to w Emulatorze przyjęto, że również w przypadku B7 o skoku decydują jedynie bity 9..21 (zerowa lub niezerowa część AR).

22) **Rozkazy niezdefiniowane**
Rozkazy o kodach nie wykorzystanych w konstrukcji maszyny, jak np. :006 czy :007, kończą się po 4 krokach przejściem do następnego rozkazu nie wykonując żadnej operacji, nawet ustawienia warunków. Są to istotnie rozkazy „NIC NIE RÓB”. Rozkazy zdefiniowane tylko potocznie określa się jako „Nic Nie Rób”, gdyż zwykle co najwyżej ustawiają warunki. Rozkaz :000, zostawiany jako ślad przez rozkaz Skoku ze Śladem, jest w istocie rozkazem arytmetycznym ustawiającym warunek Z (zero). Uniemożliwia to podprogramowi powrót po śladzie z wynikiem w postaci ustawionego warunku. Rozwiązanie problemu to zostawianie adresu powrotu w rejestrze B6 i, w razie potrzeby, powrót rozkazem niezdefiniowanym. Za najbardziej odpowiedni do tego celu uważam rozkaz niezdefiniowany :766 (Nic).

23) **Ustawianie warunków**
W ODRZE 1003/1013 rolę rozkazów porównywania pełnią rozkazy odejmowania, których wynik nie musi być gdziekolwiek posyłany, ale ustawiają w naturalny, prawidłowy sposób warunki U i Z nawet w przypadku nadmiaru. Wynik wielu innych rozkazów arytmetycznych też nie musi być gdziekolwiek posyłany, co uogólnia tę rolę, inaczej niż w wielu innych komputerach. Wynik posłany do rejestru lub pamięci i dopiero potem testowany, nie dość że utraci informację o nadmiarze, to również warunki U i Z mogą być ustawione odmiennie niż bezpośrednio po rozkazie odejmowania.

24) **Nadmiar zmiennoprzecinkowy**
W przypadku nadmiaru NZ wyniki operacji zmiennoprzecinkowych mogą różnić się od wyników rzeczywistej maszyny. Np. gdy podczas dodawania zmiennoprzecinkowego wystąpi nadmiar zmiennoprzecinkowy, to mantysa jest obliczona, co niekoniecznie musi być zgodne z rzeczywistą maszyną.
Co prawda należy unikać wystąpienia nadmiaru NZ, bowiem po jego wystąpieniu maszyna zatrzymuje się, ale możliwe jest wznowienie wykonywania się programu z niezgodną wartością.

25) **Oznaczenia, Symbole i Mnemoniki rozkazów**:

- Rolę akumulatora zmiennoprzecinkowego A* pełni para rejestrów: akumulator mantysy Am (czyli A) i akumulator cechy Ac. Oznaczenie A* zmieniłem na AC (analogicznie do AM), by znak gwiazdki (krzyża maltańskiego ✠) pozostawić dla zapisu operacji arytmetycznych.
- Symbole ∧, ÷ i · (koniunkcji, różnicy symetrycznej i mnożenia) używane w matematyce, zmieniłem odpowiednio na programistyczne &, ^ i * stosowane w powszechnie znanych językach programowania.
- Oznaczenie argumentu w komórce pamięci o adresie N zmieniłem z małej litery n na [N], by zapis funkcji rozkazu był jaśniejszy.
- Mnemoniki rozkazów we/wy zmieniłem na We0 (czytnik), We1 (dalekopis), We2 (czytnik), Wy5 (perforator) i Wy6 (dalekopis), tak by cyfra była zgodna z triadą T w kodzie operacji TPG, wyznaczającą urządzenie.
- Mnemoniki rozkazów we/wy 8-kanałowego zaczynają się od znaku X, a dalej są takie same jak 5-kanałowego: XWe0 (czytnik), XWe2 (czytnik), XWy5 (perforator/samopis), dzięki czemu nie zawierają w sobie znaku specjalnego (').
- Mnemoniki rozkazów Lw i Pr zmieniłem odpowiednio na LL i LP (Logicznie w Lewo i Logicznie w Prawo), by uniknąć niepewności „logiczne czy arytmetyczne to przesunięcia”.
- Mnemoniki rozkazów KC- i KC+ zmieniłem na SkLC (Skok z Licznikiem Cykli) z argumentem odpowiednio B-- i B++, tak by (jak we wszystkich rozkazach skoków) mówiły o skoku pod adres N (a nie pod K, który jest jak zwykle adresem następnego rozkazu).
- Mnemonik rozkazu SkNd (skoku przy nadmiarze stałoprzecinkowym) zmieniłem na SkV. Oznaczyłem nadmiar stałoprzecinkowy literą V, by uzyskać większą przejrzystość opisów rozkazów, uniknąć potencjalnego konfliktu mnemoników i zarezerwować literę N dla oznaczania przeciwnego warunku, gdyż warunkiem przeciwnym do D (dodatnie) jest ND (nie dodatnie), więc SkND oznaczałby Skok przy NieDodatnim.
- Rozkazowi Skoku przy Braku Gotowości urządzenia wejściowego nadałem mnemonik SkBG z argumentem B, gdzie B jest wartością pola MR (B) rozkazu, wskazującą numer urządzenia (czytnik 0, dalekopis 1, czytnik 2, ...).
  W rozkazach we/wy numer urządzenia jest zakodowany w kodzie TPG operacji, zaś pole B wskazuje rejestr będący argumentem lub służący do B-modyfikacji rozkazu.
  W rozkazie SkBG rejestr nie jest argumentem, ale może służyć do B-modyfikacji, a jednocześnie jego numer wskazuje urządzenie. Wskazywanie urządzenia poprzez numer rejestru B, a nie poprzez kod operacji, stanowi odstępstwo od sposobu wskazywania urządzeń w innych rozkazach. Wskutek tego numer rejestru modyfikującego rozkaz SkBG musi być zgodny z numerem urządzenia, którego gotowość sprawdzamy. Podkreśleniu tego służy wydzielenie numeru urządzenia w argument mający postać rejestru.

- Mnemoniki rozkazów BF1, BF2, FB1 i FB2 zmieniłem na BF61, BF62, FB61 i FB62, by zawierały w sobie numer (dziesiętny) ścieżki bębnowej zastępowanej przez pamięć ferrytową.
- Maszyna ma 41 rozkazów niezdefiniowanych, które nie wykonują żadnej operacji, nawet ustawienia warunków, a jedynie przechodzą do następnego rozkazu zgodnie z adresem K. Ponieważ rozkaz taki może być przydatny np. do powrotu z podprogramu ustawiającego warunek, wyznaczyłem spośród nich jeden kod, :766, jako najbardziej odpowiedni do tego celu i nadałem mu mnemonik Nic – tj. uznałem kod :766 za zdefiniowany.

26) **Kopiowanie offline na samopis**
Samopis MAW nie ma własnego czytnika; drukowanie obrazu z tasiemki możliwe jest tylko za pomocą prostego programu na ODRĘ.

27) **Wstrzymywanie wyjścia**
Urządzenia wyjściowe (perforatory, drukarka dalekopisu, samopis) nie mają przycisku (S), lecz przycisk (O) włączania Online/Offline. Komputer nie wie, czy urządzenie jest włączone, czy wyłączone (czy odebrało dane, czy nie). Wyłączenie przyciskiem (O) powoduje, że dane posyłane przez komputer trafiają w nicość. Wstrzymać, np. drukowanie, bez utraty wyników można tylko wstrzymując program, np. przyciskiem [StopCPU] lub zatrzymując niezbędne urządzenie wejściowe, albo spowolnić emulator do biegu wymagającego ponaglenia.

28) **Przełączanie wejścia z dalekopisu**
Klawiatura dalekopisu w trybie PC, oraz tekst podłączany jako „wyręka” klawiatury, mogą w miejsce jednego znaku wejściowego generować kilka kodów ITA (np. klawisz literowy A po cyfrowym generuje kody LS i A). Jednakże komputerowe rozkazy czytania wczytują po jednym kodzie ITA. Po wczytaniu pierwszego kodu (LS) można przełączyć dalekopis na inne wejście (tekst lub czytnik), a wtedy nastąpi skasowanie drugiego, nie wczytanego kodu (A), po to by po powrotnym przełączeniu wejścia nie nastąpiło ambarasujące wczytanie kodu z bufora zamiast z kolejnego znaku tekstu lub oczekiwania na znak z klawiatury.

29) **Wejście z pliku tekstowego**
Jeśli dalekopis znajduje się w trybie małych liter, to trójznakowa sekwencja wielkich liter w pliku tekstowym przełącza dalekopis w tryb wielkich liter, tak jak to robi klawisz CapsLock klawiatury.

30) **Sterowanie wysuwem papieru dalekopisu**
Wydruk z dalekopisu jest emulowany w postaci pliku tekstowego, w którym kod CR nie może cofnąć karetki do początku bieżącego wiersza. W rzeczywistym dalekopisie można było wielokrotnie pisać w tym samym wierszu, uzyskując efekt przenikania się znaków. Podobnie nie symuluje się pozostawienia karetki w bieżącej kolumnie po wysłaniu kodu LF wysuwu papieru.
W emulatorze kod LF wysuwa papier i automatycznie cofa karetkę do początku wiersza, zaś kod CR nie zmienia pozycji na wydruku, ale powoduje zresetowanie licznika znaków w wierszu tak, jak gdyby następny znak miał znaleźć się na początku wiersza, co może wpłynąć na liczbę spacji generowanych przez znak tabulacji pochodzący z fizycznej klawiatury komputera-hosta lub pliku tekstowego wyręczającego klawiaturę. Poza tym efektem kod CR jest obojętny – niemożliwe jest więc podkreślanie i przekreślanie tekstu, czy dodawanie akcentów do liter, jak np.: wiersz pierwszy ĄĆĘŁŃOŚŹŻ. W wynikowym pliku tekstowym wydruku, kod LF jest zamieniany na parę kodów (CR, LF) w systemach Windows – w pozostałych występuje w pojedynkę.
Mimo iż kod CR w emulatorze jest obojętny (czyli zbędny), to do przechodzenia do nowego wiersza zaleca się konsekwentne stosowanie sekwencji (CR, LF), a nie samotnego kodu LF, by programy pozostawały zgodne z działaniem rzeczywistych dalekopisów.

31) **Wypuszczanie NU na perforator**
Zakładając, że tak właśnie działały oryginalne perforatory, przycisk (⋮) wypuszcza kody NU na tasiemkę perforowaną również podczas biegu CPU, czym może zakłócić wyniki.

32) **Szybkość komputera**
Czas nie jest emulowany dokładnie.

33) **Dźwięki**
Emulator ma głośnik będący monitorem akustycznym, tj. służący do zwracania uwagi operatora na zatrzymanie się maszyny. Efekty dźwiękowe zależne od szybkości pracy programu nie są emulowane.

Dodatek

Pierwszy algorytm Robertsona mnożenia liczb

Mnożenie stałoprzecinkowe `AM = A * M;` w ODRZE odbywa się na liczbach:
mnożna – w rejestrze A, powstała podczas wykonywania rozkazu w fazie sumowania,
mnożnik – w rejestrze M, załadowanym argumentem uczestniczącym też w fazie sumowania,
iloczyn – powstaje w rejestrze długim AM, będącym parą rejestrów A (bity 0..38) i M (bity 1..38).

Przez A, M, R, x, s rozumiemy tu rejestry wraz z ich bitami technicznymi	
`M = M >>> 1;`	Przesunięcie logiczne mnożnika o 1 bit w prawo. Najmłodszy bit mnożnika trafia na bit techniczny rejestru M, a bit znaku rejestru jest zerowany.
`R = A >> 1;`	Przepisanie mnożnej do rejestru rozkazów z przesunięciem arytmetycznym o 1 bit w prawo. Najmłodszy bit mnożnej trafia na bit techniczny rejestru R. Pojawiający się dziwny rozkaz w R nie jest rozkazem, lecz liczbą.
`A = 0;`	Do AM na coraz to starszych pozycjach będzie dodawana wartość mnożnej R. W tym celu wystarczy dodawać R do A i przesuwać AM w prawo. W ten sposób para AM będzie zawierać coraz to nowy iloczyn częściowy wraz z potrzebnymi jeszcze bitami mnożnika.
`for (let i=0; i<38; i++) {`	W 38 półkrokach, dla kolejnych od najmłodszego do najstarszego bitów mnożnika, wykonuje się sumowania i przesunięcia w prawo otrzymywanego iloczynu częściowego wraz z mnożnikiem.
`x = M39 ? R : 0;` `A = s = A + x;`	Jeśli najmłodszy bit mnożnika jest jedynką, to do iloczynu częściowego należy dodać mnożną. Wystarczy dodać ją do części znajdującej się w A.
`Ω = A39;`	Najmłodszy bit (techniczny) rejestru A staje się w tym półkroku bitem zaokrągleń.
`A = A >> 1;` `M = (M >>> 1) + (Ω << 38);`	Najmłodszy bit mnożnika w M nie jest już potrzebny, więc następuje przesunięcie AM o 1 bit w prawo z ominięciem bitu znaku rejestru M. Następne dodawanie mnożnej do A będzie się więc odbywało na pozycjach o 1 bit starszych, dla kolejnego od końca bitu mnożnika.
`}`	Po 38 sumowaniach iloczyn jest prawie gotowy. Na bicie technicznym rejestru M znalazł się bit znaku mnożnika.

`x = M39 ? R : 0;`
`A = s = A - x;`
`M39 = 0;`
	Wystarczy teraz skorygować wynik – odjąć mnożną jeśli mnożnik był ujemny.
	Należy też skasować bit techniczny w M.

`A = A << 1;`
`Nd = (mnożna == -1) &&`
`(mnożnik == -1);`
`U = (A < 0);`
`Z = (AM == 0);`
	Trzeba również zejść z bitu technicznego w A,
	oraz ustawić rejestry warunków.

Tabelka pokazuje operacje wykonywane przez emulator w poszczególnych półkrokach.

Algorytm traktuje argumenty A i M jako liczby ułamkowe z kropką binarną tuż po bicie znaku. Wynik AM jest wtedy też liczbą ułamkową z kropką binarną tuż po bicie znaku, i ma 76 bitów ułamkowych. Nadmiar wystąpi tylko w przypadku, gdy oba czynniki mają wartość -1 w skali 0.

Argumenty można uważać za liczby w innej skali. Jeśli przyjmiemy dla nich np. skalę 38, to iloczyn będzie liczbą w skali 76, czyli, ponieważ bit 0 rejestru M nie wlicza się do liczby w AM, bit jedności iloczynu znajdzie się na bicie 77 pary AM, tj. na bicie 38 rejestru M.

Mnożenie zmiennoprzecinkowe `AC = AC * x;` w ODRZE odbywa się na liczbach:

mnożna – w rejestrze AC, czyli mantysa w rejestrze Am (tj. A), a cecha w rejestrze Ac,
mnożnik – słowo x, z mantysą na bitach 0..31 i cechą na bitach 32..38,
iloczyn – cecha w rejestrze Ac
– mantysa w rejestrze długim AM (bity $A_{0..38}$ i $M_{1..31}$),
ale podmiar, normalizacja, zaokrąglenie i warunki biorą pod uwagę tylko bity $A_{0..38}$,
a po zaokrągleniu logicznym pozostają tylko bity $A_{0..31}$.

Przez A, M, R, x, s rozumiemy tu rejestry wraz z ich bitami technicznymi	
`s = (Ac + ((x<<<32) >> 33) + 1) << 1;` `//` z uwzględnieniem bitu znaku cechy x	Suma cech czynników – z uwzględnieniem przesunięcia mantysy mnożnika o 1 bit w prawo dla uniknięcia nadmiaru przy mnożeniu mantys – pojawia się już w kroku 4.

`Ac = s >> 1;`
`if (Ac < -64) {A = 0; Ac = -64;}`
`if (Ac > 63) {NZ = 1; Ac = Ac & 127;}`
	Suma cech do rejestru Ac.
	Uwzględnienie podmiaru – zero zmiennoprzecinkowe.
	Uwzględnienie nadmiaru – obcięcie cechy.

`M = (x >> 1) >>> 7;`	Przeskalowanie prewencyjne mantysy mnożnika o 1 bit i przesunięcie logiczne w prawo na bit techniczny M.
`R = A >> 1;`	Przepisanie mantysy mnożnej do rejestru rozkazów z przesunięciem arytmetycznym na bit techniczny R.
`A = 0;`	Mnożenie mantys jest takie samo jak w stałym przecinku.

`for (let i=0; i<32; i++) {`
`  x = M39 ? R : 0;`
`  A = s = A + x;`
`  Ω = A39;`
`  A = A >> 1;`
`  M = (M >>> 1) + (Ω << 38);`
`}`
	Odbywa się w 32 półkrokach sumowań, bowiem mantysa mnożnika ma, po prewencyjnym przeskalowaniu, jedynie 32 bity po bicie znaku.

`x = M39 ? R : 0;`
`A = s = A - x;`
`M39 = 0;`
	Wystarczy teraz skorygować wynik – odjąć mantysę mnożnej jeśli mnożnik był ujemny.
	Należy też skasować bit techniczny w M.

`A = A << 1;`	Trzeba również zejść z bitu technicznego w A.
`if (A == 0) Ac = -64;`	Jeśli mantysa zerowa, przyjąć zero zmiennoprzecinkowe.
`while (A0 == A1) {A <<= 1; Ac--;}` `if (Ac < -64) {A = 0; Ac = -64;}`	Jeśli należy, to wykonać normalizację (z pominięciem M). W jej wyniku może pojawić się zero zmiennoprzecinkowe.
`A = (A | (A32 << 1)) & -256;`	Jeśli należy, to wykonać zaokrąglenie logiczne.
`U = (A < 0);` `Z = (A == 0);`	Ustawić pozostałe rejestry warunków.

Tabelka pokazuje operacje wykonywane przez emulator w poszczególnych półkrokach.

Nierestytucyjna metoda dzielenia liczb

Dzielenie stałoprzecinkowe `A = A / M;` w ODRZE odbywa się na liczbach:

dzielna – w rejestrze A, powstała podczas wykonywania rozkazu w fazie sumowania,

dzielnik – w rejestrze M, załadowanym argumentem uczestniczącym też w fazie sumowania,

iloraz – powstaje w rejestrze A (bity 0..38) i w rejestrze zaokrągleń Ω (bit numer 39).

Przez A, M, R, x, s rozumiemy tu rejestry wraz z ich bitami technicznymi	

`M = x;`
`if (M == 0)  { Nd = 1; STOP; }`
`if (będzie nadmiar)  { Nd = 1;`
`  if (StopNd) STOP;`
`  else  pomijanie dzielenia;`
`}`
	Dzielnik w M pojawia się już w kroku 4 i nie zmienia się.
	Wykrycie dzielenia przez zero – już w kroku 4.
	Wykrycie nadmiaru – już w kroku 4.
	
	Dzielenie zawsze trwa 42 kroki, nawet przy nadmiarze.

`R = A;`
 	
`A = 0;`
 	
`if (R0 == M0)  Ω = 1;`
`else  Ω = 0;`
	W rejestrze rozkazów R będą obliczane kolejne reszty częściowe – na początku jest to podwojona wartość dzielnej przesuniętej na bit techniczny.
	
	W rejestrze zaokrągleń Ω oblicza się kolejne bity ilorazu, zależne od bitów znaku reszty R i dzielnika M.

`for (let i=0; i<40; i++) {`
 	
`  if (R == 0)`
`  Ω = 0;`
`  else {`
 	
`  if (Ω == 1) R = R - M;`
`  else  R = R + M;`
 	
`  if ((R0 == M0) || (R == 0))`
`  Ω = 1;`
`  else`
`  Ω = 0;`
 	
`  R = R <<< 1;`
`  }`
 	
`  A = A <<< 1;`
`  A39 = Ω;`
 	
`}`
	W 40 krokach oblicza się 40 bitów ilorazu w A, przesuwając A w lewo.
	
	Gdy ostatnio powstała reszta jest zerowa, to teraz już do końca przyjmuje się bity ilorazu równe 0.
	
	Zależnie od zgodności znaków reszty i dzielnika oblicza się nową resztę częściową przez odjęcie lub dodanie dzielnika do wcześniej podwojonej reszty częściowej.
	
	Gdy nowa reszta częściowa jest zerowa, to przyjmuje się nowy bit ilorazu równy 1, a wszystkie następne równe 0; w przeciwnym razie nowy bit ilorazu zależy od zgodności znaków reszty i dzielnika.
	
	Resztę częściową podwaja się przesunięciem, tak że jest przygotowana do kolejnego dodania lub odjęcia dzielnika.
	
	Zawartość A przesuwa się o 1 bit w lewo i na najmłodszą pozycję (na bit techniczny) wstawia obliczony bit ilorazu.
	
	Po 40 sumowaniach iloraz jest prawie gotowy.

`A39 = 0;`
 	
`U = (A < 0);`
`Z = (A == 0);}`
	Trzeba usunąć z A zbędny bit techniczny – jest w Ω.
	
	oraz ustawić pozostałe rejestry warunków.

Tabelka pokazuje operacje wykonywane przez emulator w poszczególnych krokach.
Należy zwrócić uwagę, że dodawanie, odejmowanie i przesunięcia odbywają się w 40-bitowych rejestrach, w których starsze bity znikają i bitem znaku staje się najstarszy zachowany bit rejestru.

Algorytm traktuje argumenty A i M jako liczby ułamkowe z kropką binarną tuż po bicie znaku. Wynik A jest wtedy też liczbą ułamkową z kropką binarną tuż po bicie znaku. Jeśli dzielna i dzielnik są takie, że iloraz nie mieści się w A w skali 0 (w postaci liczby z kropką binarną tuż po bicie znaku), to dzielenie nie jest wykonywane. Jeśli dzielnik jest zerem, to dzielenie również nie jest wykonywane i maszyna zatrzyma się.

Argumenty można uważać za liczby w innej skali, wtedy iloraz powstaje w skali umownej równej różnicy skal dzielnej i dzielnika. Warunkiem wykonalności dzielenia jest, by dzielna była mniejsza co do wartości bezwzględnej od dzielnika (równa tylko gdy mają różne znaki).

Dzielenie stałoprzecinkowe długie `A = A // M, N;` w ODRZE odbywa się na liczbach:
dzielna – w rejestrze A, załadowanym przez wcześniejsze rozkazy,
dzielnik – w rejestrze M, załadowanym przez wcześniejsze rozkazy,
iloraz – powstaje w rejestrze A (bity 0..38) i w rejestrze zaokrągleń Ω (bit numer 39).

Dzielenie długie jest inicjowane rozkazem `DzD N`, i wykonywane tak samo jak zwykłe dzielenie, ale w którym liczba wykonań pętli „for” wynosi nie 40, a N-3. Najstarsze bity ilorazu nie mieszczące się w rejestrze A są tracone, więc w wyniku otrzymuje się ostatnie 40 bitów (39 bitów w akumulatorze A, oraz 40. bit w rejestrze zaokrągleń Ω). I tak w szczególności:
- dla N=43=40+3, na bitach akumulatora $A_0..A_{38}$ znajdą się bity ilorazu $i_0..i_{38}$, oraz bit i_{39} w Ω,
- dla N=82=39+43, na bitach akumulatora $A_0..A_{38}$ znajdą się bity ilorazu $i_{39}..i_{77}$, oraz bit i_{78} w Ω,
- dla N=121=39+39+43, na bitach $A_0..A_{38}$ znajdą się bity ilorazu $i_{78}..i_{116}$, oraz bit i_{117} w Ω,
- dla N=160=39+39+39+43, na $A_0..A_{38}$ znajdą się bity ilorazu $i_{117}..i_{155}$, oraz i_{156} w Ω,
- dla N=199=39+39+39+39+43, na $A_0..A_{38}$ znajdą się bity ilorazu $i_{156}..i_{194}$, oraz i_{195} w Ω,
- dla N=238=39+39+39+39+39+43, na $A_0..A_{38}$ znajdą się bity ilorazu $i_{195}..i_{233}$, oraz i_{234} w Ω,
- dla N=255=39+39+39+39+39+39+21, na $A_0..A_{38}$ znajdą się bity ilorazu $i_{212}..i_{250}$, oraz i_{251} w Ω.
  UWAGA: Bity $A_0..A_{21}$ przy N=255 pokrywają się z bitami $A_{17}..A_{38}$ przy N=238.

W przypadkach zdegenerowanych, gdy N < 43, wynik jest następujący:
- dla N<2 dzielenie w ogóle nie jest rozpoczynane,
- dla N=2 zostanie wyzerowany rejestr A i obliczony wstępnie bit Ω,
- dla N=3 zostanie wyzerowany rejestr A, obliczony wstępnie bit Ω, oraz ustawione warunki,
- dla N=4 zostanie wyzerowany rejestr A, obliczony bit ilorazu Ω, oraz ustawione warunki,
- dla N=5 w A zostanie obliczony bit A_{38} poprzedzony zerami, bit Ω, oraz ustawione warunki,
- dla N=6 w A zostaną obliczone 2 bity na pozycjach A_{37} i A_{38}, bit Ω, oraz ustawione warunki,
- dla N=7 w A zostaną obliczone 3 bity na pozycjach $A_{36}..A_{38}$, bit Ω, oraz ustawione warunki,
- itd.

Dzielenie zmiennoprzecinkowe `AC = AC / M;` w ODRZE odbywa się na liczbach:
dzielna – w rejestrze AC, czyli mantysa w rejestrze Am (tj. A), a cecha w rejestrze Ac,
dzielnik – słowo x, z mantysą xm na bitach 0..31 i cechą xc na bitach 32..38,
iloraz – cecha w rejestrze Ac
– mantysa w rejestrze Am (czyli w A, na bitach $A_{0..38}$, a po zaokrągleniu tylko na $A_{0..31}$).

Przez A, M, R, x, s rozumiemy tu rejestry wraz z ich bitami technicznymi	

`s = (Ac - ((x<<<32) >> 33) + 1) << 1;`
`// z uwzględnieniem bitu znaku cechy x`
`M = x - (x & 255);`
`if (M == 0) { Nd = 1; STOP; }`
`if (będzie nadmiar) { Nd = 1;`
`if (StopNd) STOP;`
`else pomijanie dzielenia;`
`}`
	Różnica cech dzielnej i dzielnika – z uwzględnieniem przesunięcia mantysy dzielnej o 1 bit w prawo dla uniknięcia nadmiaru przy dzieleniu mantys – pojawia się już w kroku 4.
	Wykrycie dzielenia przez zero i nadmiaru – już w kroku 4.
	Dzielenie zmiennoprzecinkowe zawsze trwa 44 kroki.

`Ac = s >> 1;`
`if (Ac < -64)  { A = 0; Ac = -64; }`
`else if (Ac > 63) { NZ = 1; STOP; }`
	Przy podmiarze liczba w rejestrze AC jest zamieniana na zero zmiennoprzecinkowe.
	Przy nadmiarze cecha w Ac jest obcinana do 7 bitów.

`R = A >> 1;`
 	
`A = 0;`
	W rejestrze rozkazów R będą obliczane kolejne reszty częściowe – na początku jest to wartość dzielnej przesuniętej na bit techniczny.

`if (`R_0 `==` M_0`)  Ω = 1;` `else  Ω = 0;`	W rejestrze zaokrągleń Ω oblicza się kolejne bity ilorazu, zależne od bitów znaku reszty R i dzielnika M.
`for (let i=0; i<40; i++) {`	W 40 krokach oblicza się 40 bitów ilorazu w A, przesuwając A w lewo.

`  if (R == 0)`
`  Ω = 0;`
`  else {`
	Gdy ostatnio powstała reszta jest zerowa, to teraz już do końca przyjmuje się bity ilorazu równe 0.

`  if (Ω == 1) R = R - M;` `  else  R = R + M;`	Zależnie od zgodności znaków reszty i dzielnika oblicza się nową resztę częściową przez odjęcie lub dodanie dzielnika do wcześniej podwojonej reszty częściowej.

`  if ((`R_0 `==` M_0`) || (R == 0))`
`  Ω = 1;`
`  else`
`  Ω = 0;`
	Gdy nowa reszta częściowa jest zerowa, to przyjmuje się nowy bit ilorazu równy 1, a wszystkie następne równe 0; w przeciwnym razie nowy bit ilorazu zależy od zgodności znaków reszty i dzielnika.

`  R = R <<< 1;` `  }`	Resztę częściową podwaja się przesunięciem, tak że jest przygotowana do kolejnego dodania lub odjęcia dzielnika.
`  A = A <<< 1;` A_{39} `= Ω;`	Zawartość A przesuwa się o 1 bit w lewo i na najmłodszą pozycję (na bit techniczny) wstawia obliczony bit ilorazu.
`}`	Po 40 sumowaniach iloraz jest prawie gotowy.
A_{39} `= 0;`	Trzeba usunąć z A zbędny bit techniczny – jest w Ω.
`if (A == 0) Ac = -64;`	Jeśli mantysa zerowa, przyjąć zero zmiennoprzecinkowe.
`while (`A_0 `==` A_1`) {A <<= 1; Ac--;}` `if (Ac < -64) {A = 0; Ac = -64;}`	Jeśli należy, to wykonać normalizację. W jej wyniku może pojawić się zero zmiennoprzecinkowe.
`A = (A	(`$A_{32}$ `<< 1)) & -256;`	Jeśli należy, to wykonać zaokrąglenie logiczne.
`U = (A < 0);` `Z = (A == 0);}`	Ustawić pozostałe rejestry warunków.

Dzielenie zmiennoprzecinkowe może być niewykonalne, gdy argumenty są nieznormalizowane. Jeśli dzielnik jest znormalizowany, to dzielenie jest wykonalne.

Jeśli dzielna jest nieznormalizowana, to mantysa ilorazu będzie miała mniej bitów znaczących.

Mantysa dzielnika pozostaje w rejestrze M aż do zakończenia wykonywania się rozkazu.

Podsumowanie

Oprogramowanie emulatora nie jest obszerne, nie wymaga instalowania czegokolwiek, jest w postaci tekstowej. Potrzebna jest jedynie przeglądarka internetowa w wystarczająco świeżej wersji, np. Firefox, Edge, Chrome. Ja testuję głównie na systemach operacyjnych Windows 10 i Linux Ubuntu, oraz na przeglądarkach internetowych Firefox, Edge i Chrome.

Klemens Czajka