Adres Redakcji:
Warszawa, Nowy Świat 72 pokój 14
Zakład Automatyki PAN
tel. 6-64-23

Redakcja czynna: poniedziałki, środy i piątki

PAŃSTWOWE WYDAWNICTWO NAUKOWE — WARSZAWA
Miodowa 10

Nakład 750 (632+118)	Oddano do składania 12.VII.58
Ark. wyd. 5,25. Ark. druk. 4,5	Podpisano do druku 22.XI.58
Papier druk. sat. kl. III, 80 g, 70 × 100	Druk ukończono w listopadzie 1958
Cena zł 25.—	Zam. 1126. A-31

DRUKARNIA IM. REWOLUCJI PAŹDZIERNIKOWEJ, WARSZAWA

WOJCIECH JAWORSKI

Programowana maszyna cyfrowa z nową budową instrukcji

Rękopis dostarczono 17.II.1958 r.

Projektowane początkowo maszyny cyfrowe posiadały odrębną pamięć instrukcji i pamięć argumentów. Następnym etapem rozwojowym to budowa maszyn ze wspólną pamięcią dla instrukcji i argumentów. W pamięci tej instrukcje i argumenty występują jednak rozłącznie. W niniejszej pracy starano się zrobić dalszy krok na tej drodze opracowując koncepcję maszyny cyfrowej z kodem instrukcyjnym, w którym argument może stanowić część instrukcji. Jednocześnie podano pewien schemat adresowania i pobierania zastępujący stosowane dotąd metody adresowania i modyfikacji instrukcji, realizowane za pomocą specjalnych urządzeń, instrukcji lub programów.

1. WSTĘP

Praca niniejsza przedstawia pewną koncepcję programowanej maszyny cyfrowej. Istotną cechą tej koncepcji jest nowa budowa instrukcji[1]. Budowa ta pociągnęła za sobą zarówno zmianę struktury maszyny cyfrowej, jak i istotne zmiany w programowaniu. Zmiany te występują nie tylko przy układaniu programów z pojedynczych instrukcji i w systemie adresowania, lecz także w korzystaniu z gotowych programów bibliotecznych (za pomocą wprowadzonej instrukcji programowanej). Instrukcja programowana łącznie z obowiązującym dla niej schematem adresowania i pobierania zapewnia znaczne uproszczenie składania programów z programów bibliotecznych oraz daje podobne ułatwienia jak technika interpretacyjna.

Wprowadzona w tej pracy lista instrukcji nie jest optymalna. W miarę potrzeby lista instrukcji może być modyfikowana i uzupełniana.

Mimo że nie będziemy rozpatrywać szczegółowych układów i rozwiązań technicznych, to jednak staraliśmy się zapewnić możliwość prostej

[1] Pomysł wprowadzenia nowej budowy instrukcji dała analiza opisanych w [21], [31], [8], [54] metod stosowanych dla przyśpieszenia liczenia w maszynach z pamięcią opóźnieniową, a zwłaszcza analiza metody programowania optymalnego oraz praca [26].

realizacji technicznej i zmniejszenia udziału czasu potrzebnego na korzystanie z pamięci w ogólnym czasie liczenia.

Proponowany kod instrukcyjny zmniejsza ilość operacji pobierania i przesyłania do pamięci (na skutek istnienia argumentów w instrukcji i instrukcji przesyłania zespołowego), co ma szczególne znaczenie dla maszyn z pamięcią o dużym czasie dostępu.

Analiza wykorzystania pamięci dała asumpt do opracowania opisanego szkicowo kodu instrukcyjnego o zmiennej długości instrukcji. Wydaje się celowe podkreślenie dużych możliwości zawartych w tym kodzie (opis szkicowy w rozdziale 5).

Podstawowe tezy pracy niniejszej były referowane dnia 25.X.1955 roku na seminarium Zakładu Aparatów Matematycznych I. M. PAN.

Miłym dla mnie obowiązkiem jest wyrażenie wdzięczności: doc. R. Marczyńskiemu za cenne uwagi i kierownictwo pracą, prof. dr. H. Greniewskiemu za pomocną dyskusję nad meritum i ujęciem tematu, kolegom z Zakładu Aparatów Matematycznych Instytutu Matematycznego a w szczególności knt nauk Z. Pawlakowi, magistrom J. Fiettowi i A. Wakuliczowi za wiele krytycznych i rzeczowych uwag.

Chciałbym również podziękować prof. dr. K. Kuratowskiemu za umożliwienie wykonania niniejszej pracy w Instytucie Matematycznym PAN.

2. KOD INSTRUKCYJNY

2.1. BUDOWA INSTRUKCJI

Każda instrukcja, z którą w naszych rozważaniach będziemy mieli do czynienia, może być przedstawiona w każdym z trzech zapisów:

1) zapis słowny (w języku potocznym),
2) „ symboliczny (w języku symbolicznym),
3) „ cyfrowy (w języku maszyny).

Ad 1). Potrzeba używania zapisu słownego nie wymaga uzasadnienia.

Ad 2). Zapis symboliczny, ze względu na swą zwięzłość, posiada dużą wartość praktyczną i jest używany przez matematyka przy programowaniu.

Ad 3). Zapis ten jest niezbędny przy konstruowaniu programowanej maszyny cyfrowej.

Zapis cyfrowy dowolnej instrukcji posiada w naszej koncepcji następującą postać:

$$\underbrace{\alpha_0 \dots \alpha_n}_{A}\ \underbrace{\beta_0 \dots \beta_n}_{B}\ \underbrace{\gamma_0 \dots \gamma_n}_{C}\ \underbrace{\delta_0 \dots \delta_n}_{D}$$

Jak widać, instrukcja składa się wyłącznie z czterech rozłącznych części A, B, C, D, nazywanych dalej „słowami". Każde słowo zawiera $n+1$ cyfr.[2] W większości przypadków słowo D wyznacza bezpośrednio operację, jaka ma być wykonana na argumentach wyznaczonych przez słowa A i B. Słowo C wyznacza miejsce wyniku.

Nasza instrukcja przypomina strukturą instrukcję w maszynie trójadresowej (np. $S\ E\ A\ C$ w połączeniu trójadresowym). Słowo D w powyższym schemacie odpowiada części operacyjnej, a słowa A, B, C adresom instrukcji trójadresowej.

Proponowany kod instrukcyjny podaje tablica 2.1.1.

Jak widać z tablicy 2.1.1. instrukcje nr 1—4 dotyczące operacji arytmetycznych na liczbach (zawierające „$\Rightarrow$" jako symbol przesłania zostały nazwane „instrukcjami pierwotnymi". Instrukcje nr 6 i 7 dotyczące operacji arytmetycznych na adresach (zawierające „$\Rightarrow$" jako symbol przesłania) nazwano „instrukcjami wtórnymi".

Instrukcje nr 5 i 9 dotyczące wyboru następnej instrukcji zostały nazwane instrukcjami warunkowymi[3].

Instrukcja pierwotna warunkowa wystąpi np. w programie obliczenia iteracyjnego, gdzie zakończenie iteracji jest uzależnione od spełnienia wyznaczonego liczbą warunku.

Instrukcja wtórna warunkowa będzie występować np. w programie mnożenia skalarnego dwóch wektorów, w którym ilość obiegów pętli jest wyznaczona adresem słowa zawierającego pierwszy składnik wektora i adresem słowa zawierającego ostatni składnik wektora.

Instrukcje wtórne oraz obie instrukcje warunkowe umożliwiają wielokrotne wykonywanie tych samych instrukcji pierwotnych na różnych liczbach i sterowanie tokiem obliczenia zależnie od otrzymanych wyników.

W instrukcji nr 8 występuje pojęcie kolejności słów. Sprecyzowanie tego pojęcia może być dokonane (w sposób odpowiadający intencjom projektanta) tylko w związku z bardziej niż tu szczegółową koncepcją organizacji i przeznaczenia maszyny (vid. przykład zawarty w tablicy 3.4.3.)[4]. Instrukcja nr 8 jest łatwa do realizacji w maszynach z pamięcią opóźnieniową (jeżeli adresy wyznaczone przez A i C mają odpowiednie współrzędne czasowe).

Instrukcja nr 10 nazwana instrukcją programowaną powoduje wywołanie sekwencji instrukcji z jednoczesnym przesłaniem informacji umo-

---

[2] W zasadzie mogą to być cyfry w dowolnym układzie, np. dziesiętnym czy binarnym.

[3] Termin wzorowany na ang. „conditional instruction .

[4] Vid. także [54] str. 263.

Tablica 2.1.1.
Kod instrukcyjny

lp.	Instrukcja: Zapis symboliczny	Instrukcja: Zapis słowny	Symbol operacji wyznaczony słowem D
I	II	III	IV
		1. INSTRUKCJE PIERWOTNE	
		1.1. Instrukcje pierwotne, bezwarunkowe	
1	$A+B\Rightarrow C$	*Dodawanie:* Dodaj liczby wyznaczone przez słowa A i B, wynik prześlij pod adres wyznaczony przez słowo C i pobierz kolejną instrukcję (o adresie zawartym w liczniku instrukcyj*)).	$+\Rightarrow$
2	$A-B\Rightarrow C$	*Odejmowanie:* Jak wyżej, zastępując dodawanie odejmowaniem.	$-\Rightarrow$
3	$A\cdot B\Rightarrow C$	*Mnożenie:* Jak wyżej, zastępując dodawanie mnożeniem.	
4	$A:B\Rightarrow C$	*Dzielenie:* Jak wyżej, zastępując dodawanie dzieleniem.	$:\Rightarrow$
		1.2. Instrukcja pierwotna, warunkowa	
5	$A\ ;\ B\overset{?}{\Rightarrow}C$	*Pobieranie:* Jeżeli różnica liczb wyznaczonych przez słowa A i B jest <0, pobierz jako kolejną instrukcję, instrukcję której adres jest wyznaczony przez słowo C. W przeciwnym razie pobierz kolejną instrukcję (o adresie zawartym w liczniku instrukcyj).	$\overset{?}{\Rightarrow}$
		2. INSTRUKCJE WTÓRNE	
		2.1. Instrukcje wtórne, bezwarunkowe	
6	$A+B\rightarrow C$	*Dodawanie:* Dodaj adresy, wyznaczone przez słowa A i B, wynik prześlij pod adres wyznaczony przez słowo C, pobierz kolejną instrukcję (o adresie zawartym w liczniku instrukcyj).	$+\rightarrow$
7	$A-B\rightarrow C$	*Odejmowanie:* Jak wyżej, zastępując dodawanie odejmowaniem.	$-\rightarrow$
8	$A\div B\rightarrow C$	*Przesyłanie zespołowe:* Prześlij kolejne słowa poczynając od słowa o adresie wyznaczonym przez słowo A, kończąc na słowie o adresie wyznaczonym przez słowo B na miejsce kolejnych słów poczynając od słowa o adresie wyznaczonym przez słowo C. Pobierz kolejną instrukcję (o adresie zawartym w liczniku instrukcyj).	$\div\rightarrow$

* Definicja wyrażenia „licznik instrukcyj" — vid. 5.2.

Tablica 2.1.1. (c.d.)

I	II	III	IV
		2.2. Instrukcja wtórna warunkowa	
9	$A\ ;\ B\overset{?}{\rightarrow}C$	*Pobieranie:* Jeżeli różnica adresów wyznaczonych przez słowa A i B jest <0, pobierz jako kolejną instruckję, instrukcję której adres jest wyznaczony przez słowo C. W przeciwnym razie pobierz kolejną instrukcję (o adresie zawartym w liczniku instrukcyj).	$\overset{?}{\rightarrow}$
		3. INSTRUKCJA PROGRAMOWANA	
10	$A;B;C;Fh$	*Przesyłanie:* Prześlij do słów o adresie wyznaczonym przez słowo D, zwiększoną o 4 zawartość licznika instrukcyj, prześlij do licznika instrukcyj zwiększony o 4 adres wyznaczony przez słowo D. Pobierz instrukcję o adresie zawartym w liczniku instrukcyj**).	Fh

** Szczegółowe działanie instrukcji programowanej wyniknie z 2.2. i 3.2.

żliwiającej powrót do instrukcji następnej po instrukcji programowanej. Adres pierwszej instrukcji wywoływanej sekwencji wyznacza słowo D[5].

Odddzielnych instrukcji wejścia i wyjścia[6] (służących do wprowadzania i wyprowadzania informacji z maszyny) nie zamieszczono, gdyż można je prosto zrealizować przez przyporządkowanie adresów wejściu i wyjściu[7].

Tablica 2.1.1. podaje możliwie prostą listę instrukcji i obejmuje tylko te instrukcje, które będą potrzebne w dalszym ciągu pracy. W instrukcjach podanych w tablicy występuje nie sprecyzowane bliżej wyrażenie *„wyznaczać"* (vid. np. w instrukcji nr 1" ... na liczbach *wyznaczonych* przez słowa A i B ..."). Okaże się wkrótce, że w powyższym kodzie instrukcyjnym wyrażenie *„wyznacza"* używane jest w sposób systematycznie wieloznaczny. Ta systematyczna wieloznaczność jest istotna dla niniejszej pracy i pozwala zrealizować następujące cele:

1. Uniezależnienie programu od położenia w pamięci
2. Uniknięcie zamiany lub podstawiania adresów dokonywanych za pomocą specjalnych instrukcji.
3. Ułatwienie wywoływania programów.

[5] Instrukcja programowana przypomina „function table order vid. [13].
[6] „input instruction", „output instruction".
[7] Vid. [2] str. 184.

Systematyczną wieloznaczność (liczb i adresów) uzyskano przez:

1) rozróżnianie czterech rodzajów adresów: adresy I rzędu, II rzędu, III rzędu i adresy instrukcji
2) odpowiednio zróżnicowane (według rodzajów instrukcji) kryteria zakończenia pobierania.

Tablica 2.1.2, oprócz wykorzystania poszczególnych cyfr słów *A*, *B*, *C*, wprowadza to rozróżnianie rodzajów adresów.

Kryteria zakończenia procesu pobierania są natomiast podane dopiero w tablicy 2.2.1.

Tablica 2.1.2.

Wykorzystanie poszczególnych cyfr w słowie X ($X = A$ albo $X = B$, albo $X = C$)

Schemat cyfrowy słowa X			
$\tau_0\tau_1 \ldots\ldots \tau_k\tau_{k+1} \ldots\ldots \tau_n$			
Wykorzystanie cyfr:			
1 0 ← adres	1-go rzędu →	← odstęp adresowy	→
1 1 ← „	II-go „ →	← „ „	→
1 2 ← „	III-go „ →	← „ „	→
1 3 ← „	instrukcji →	← „ „	→
0 ← zapis cyfrowy liczby →			

Z tabl. 2.1.2 widać, że τ_0 decyduje, czy dane słowo przedstawia sobą liczbę, czy adres ($\tau_0 = 0$ — liczba; $\tau_0 = 1$ — adres). Jeżeli $\tau_0 = 1$, to τ_1 wyróżnia rodzaj adresu utworzonego z cyfr $\tau_2 \ldots\ldots \tau_k$. Przeznaczenie cyfr $\tau_{k+1} \ldots\ldots \tau_n$ (odstęp adresowy) będzie objaśnione w 3.1.

Tablica 2.1.3.

Wykorzystanie poszczególnych cyfr w słowie *D*.

Schemat cyfrowy słowa D:			
$\delta_0\delta_1 \ldots\ldots \delta_k\delta_{k+1} \ldots\ldots \delta_n$			
Wykorzystanie cyfr:			
0 ← dowolna z instrukcji nr 1—9 →			
0 0 ← instrukcja	pierwotna	bezwarunkowa →	
0 1 ← „	„	warunkowa →	
0 2 ← „	wtórna	bezwarunkowa →	
0 3 ← „	„	warunkowa →	
1 ← „	programowana →		
1 0 ← adres	I-go rzędu →	← odstęp adresowy	→
1 1 ← „	II-go rzędu →	← „ „	→
1 2 ← „	III-go rzędu →	← „ „	→
1 3 ← „	instrukcji →	← „ „	→

Tablica 2.1.3. pokazuje wykorzystanie poszczególnych cyfr w słowie *D* każdej instrukcji. Cyfra δ_0 słowa *D* sygnalizuje, czy słowo to stanowi część jednej z instrukcji nr 1—9, czy też część instrukcji programowanej ($\delta_0 = 0$ — jedna z instrukcji nr $1 \div 9$; $\delta_0 = 1$ — instrukcja programowana).

Porównując tablicę 2.1.2 z tablicą 2.1.3 widać, że w instrukcji programowanej cyfry $\delta_1 \ldots \delta_n$ słowa *D* są wykorzystane identycznie jak cyfry $\tau_1 \ldots \tau_n$ słowa *X*, gdy $\tau_0 = 1$. Każdy symbol operacji (tablica 2.1.1. kol. IV) posiada swój odpowiednik w zapisie cyfrowym (cyfry słowa *D*), odpowiedniość ta częściowo jest podana w tablicy 2.1.4.

Tablica 2.1.4

Odpowiedniość między zapisem symbolicznym i cyfrowym symbolu operacji

Zapis symboliczny	Zapis cyfrowy (σ_0 σ_1)
I.	II.
$\Rightarrow$	0 0
$\overset{?}{\Rightarrow}$	0 1
$\rightarrow$	0 2
$\overset{?}{\rightarrow}$	0 3
Fh	1.

2.2. POBIERANIE

W 2.1. ustaliliśmy, że w zapisie cyfrowym instrukcja składa się zawsze z czterech słów: *A*, *B*, *C* i *D* i że w instrukcjach nr 1—9 słowo *D* wyznacza działanie, które ma być wykonane na słowach wyznaczonych przez słowa: *A*, *B*, *C*. (Inaczej, jak zobaczymy, przedstawia się sprawa dla instrukcji programowanej).

Wykonanie każdej z instrukcji nr $1 \div 9$ (tablica 2.1.1) ma przebieg następujący:

1. Pobranie słów na których będzie wykonane działanie (np. arytmetyczne),
2. Przygotowanie adresu wyniku,
3. Wykonanie działania (np. arytmetycznego),
4. Przesłanie wyniku,
5. Pobranie następnej do wykonania instrukcji (wyznaczonej zawartością licznika instrukcyj).

Pobieranie słów, na których maszyna wykonuje działanie, odbywa się dla wszystkich słów instrukcji nr 1—10 według jednego z dwu schematów:

1. Schematu przekształcania,
2. Schematu pobierania.

Schemat przekształcania przedstawia się następująco:

Niech $t_o \ldots t_n$ będzie słowem o adresie $\sigma_2 \ldots \sigma_k$, gdzie $\sigma_o \cdot \ldots \sigma_n$ są kolejnymi cyframi dowolnego słowa instrukcji znajdującego się w rejestrze pomocniczym (vid. 2.3). Układ pobierający słowa jest tak skonstruowany, że po wykonaniu jednego pobrania w danym słowie instrukcji znajdą się cyfry sumy liczby $(\sigma_{k+1} \ldots \sigma_n)$ i liczby $(t_2 \ldots t_k)$, zaś na miejscu cyfr $\sigma_{k+1} \ldots \sigma_n$ znajdą się cyfry $t_{k+1} \ldots t_n$. Cyfry $t_o\, t_1$ zajmą miejsce cyfr $\sigma_o\, \sigma_1$.

Schemat podstawiania polega na tym, że słowo $\sigma_o \ldots \sigma_n$, znajdujące się aktualnie w rejestrze pomocniczym, zostaje zastąpione przez słowo $t_o \ldots t_n$, znajdujące się w pamięci pod adresem $\sigma_2 \ldots \sigma_k$. Oba schematy podano jeszcze w tablicy 2.2.0.

Powyższy opis schematu przekształcania wyjaśnia zarazem rolę, jaką spełnia w adresowaniu odstęp adresowy (vid. tablica 2.1.2 i tablica 2.1.3). Łatwo też zauważyć, że jeżeli odstęp adresowy zawarty w słowie znajdującym się w rejestrze pomocniczym jest równy zeru, czyli

$$\sigma_2 = 0 \quad \text{dla } l = k + 1, \ldots n,$$

to wówczas zastosowanie schematu przekształcania daje ten sam wynik, co zastosowanie schematu podstawiania.

Pobieranie powtarza się tylokrotnie, aż zostanie pobrane odpowiednie słowo. Słowo, na którym kończy się pobieranie, wyznaczone bezpośrednio przez pierwsze dwie cyfry tego słowa, podaje dla różnych grup instrukcji tablica 2.2.1. Jak widać, warunek zakończenia pobierania jest:

1) taki sam jak dla słów A i B,
2) dla słowa C odmienny niż dla wszystkich pozostałych,
3) również dla słowa D odmienny niż dla wszystkich pozostałych.

Warunek ten zależy również od rodzaju instrukcji (od słowa D).

Podkreślić jeszcze należy, że schemat podstawiania bywa stosowany jedynie w *ostatnim* kroku pobierania i to nie zawsze (vid. tabl. 2.2.1), natomiast we wcześniejszych krokach pobierania stosuje się wyłącznie schemat przekształcania.

Jak widać z tablicy 2.2.1., proces pobierania może być ograniczony do jednego kroku (np. dla słowa A instrukcji, w którym pierwsza cyfra jest zerem, to jest $a_o = 0$), albo też wielokrokowy. W procesie wielokrokowym pobierania danego słowa danej instrukcji możemy mieć do czynienia kolejno albo z obniżeniem rzędu adresu, albo z zachowaniem rzędu adresu (sytuacja wyjątkowa, vid, instrukcja „stop", § 3.1.), albo nawet z podwyższeniem rzędu adresu.

Tablica 2.2.0

Przebieg poszczególnego pobrania dla słowa X instrukcji danej (X = A, B, C, D)

Słowo	Schemat przekształcania	Schemat podstawiania
I	II	III
Stan początkowy słowa X (w rejestrze pomocniczym)	$\sigma_o\ \underbrace{\sigma_1\sigma_2 \ldots \sigma_k}_{\substack{\text{Adres rzędu} \\ \sigma_1 + 1 \text{ ew. adres} \\ \text{instrukcji}}} \quad \underbrace{\sigma_{k+1} \ldots \sigma_n}_{\substack{\longrightarrow \\ \text{odstęp} \\ \text{adresowy}}}$	$\sigma_o\ \underbrace{\sigma_1\sigma_2 \ldots \sigma_k}_{\substack{\text{Adres rzędu} \\ \sigma_1 + 1 \text{ ew. adres} \\ \text{instrukcji}}} \quad \underbrace{\sigma_{k+1} \ldots \sigma_n}_{\substack{\longrightarrow \\ \text{odstęp} \\ \text{adresowy}}}$
Słowo znajdujące się w pamięci pod adresem $\sigma_2 \ldots\ldots \sigma_k$	$t_o\, t_1\, t_2 \ldots t_k\, t_{k+1} \ldots t_n$	$t_o\, t_1\, t_2 \ldots t_k\, t_{k+1} \ldots t_n$
Stan końcowy słowa (w rejestrze pomocniczym)	$t_o\, t_1\, t_2\, \eta_2 \ldots \eta_k\, t_{k+1} \ldots t_n$ gdzie $(\eta_2 \ldots \eta_k) = (\sigma_{k+1} \ldots \sigma_n) + (t_2 \ldots t_k)$[8]	$t_o\, t_1\, t_2 \ldots t_k\, t_{k+1} \ldots t_n$

[8] $(\eta_2 \ldots \eta_k)$, $(\sigma_{k+1} \ldots \sigma_n)$, $(t_2 \ldots t_k)$ należy traktować jako liczby zapisane cyfrowo w obowiązującym dla danej maszyny systemie, np. binarnym czy dziesiętnym; znak "+" jest tu symbolem dodawania liczb w danym systemie.

W opisanym kodzie instrukcyjnym, jak wiemy, mamy do czynienia z instrukcjami modyfikowanymi w czasie procesów przekształcania lub podstawiania. Dopiero po zakończeniu procesu pobierania następuje wykonanie działania.

Instrukcje kodu instrukcyjnego (tablica 2.1.1) mogą wyznaczyć w sposób pośredni[9] argumenty, na których ma być wykonane działanie. Wynika to ze zróżnicowania rzędów adresów (tabl. 2.1.3) i opisanego już przebiegu pobierania. Jako wynik pobierania słów instrukcji otrzymujemy bezpośrednie wyznaczenie odnośnych argumentów. Końcowym efektem pobierania słowa A oraz słowa B będzie otrzymanie bezpośrednio obu argumentów, na których ma być dokonane działanie, opisane w danej instrukcji. Podobnie końcowym efektem pobierania słowa C będzie otrzymanie bezpośredniego adresu, pod którym ma być przesłany wynik.

W tablicy 2.2.2. w sposób zbiorczy pokazano poszczególne słowa instrukcji w chwili zakończenia pobierania.

Z tablic 2.2.1. i 2.2.2. widać, że w przyjętym systemie adresowania instrukcja może być określona przez adres instrukcji lub adres I rzędu, pierwszego słowa tej instrukcji.

2.3. UPROSZCZONY SCHEMAT BLOKOWY MASZYNY CYFROWEJ

Opisaliśmy już pokrótce przebieg pobierania. Obecnie zajmiemy się przebiegiem wykonywania instrukcji przez maszynę. Będzie nas interesować zwłaszcza podział czynności między poszczególne zespoły maszyny. Wobec tego wydaje się celowe zaznajomienie się w tym miejscu z najbardziej blokowo ujętym schematem maszyny cyfrowej, pracującej według omówionego już przez nas kodu instrukcyjnego (bardziej szczegółowy schemat maszyny jest opisany w 4.2). Schemat przedstawiono na rys. 1. Maszyna składa się z pamięci P, rejestru pomocniczego RP, arytmometru AR, układu pobierająco-przesyłającego UPP i licznika instrukcyj L.

Licznik instrukcyj poprzez układ pobierająco-przesyłający steruje pobieraniem z pamięci kolejnych słów tworzących kolejną instrukcję i przesyłaniem do rejestru pomocniczego. W rejestrze pomocniczym, zaczynając od słowa D, następuje analiza kolejnych słów instrukcji.

Jeżeli $\delta_0 = 0$ (jedna z instrukcji nr 1÷9), to zachodzi pobieranie słów A,B,C bieżącej instrukcji i postępowanie z nimi wg opisu w 2.2. W przypadku, gdy dowolne z pobieranych bieżąco słów A,B,C spełnia kryteria zakończenia pobierania wymienione w tablicy 2.2.1, następuje podstawienie tego słowa bez modyfikacji do rejestru pomocniczego. Gdy każde ze słów A,B,C spełni odpowiadające danemu słowu kryterium zakończe-

[9] Z wyjątkiem instrukcji programowanej, dla której adres następnej instrukcji może być wyznaczony pośrednio.

Tablica 2.2.1.
Zakończenie pobierania poszczególnego słowa

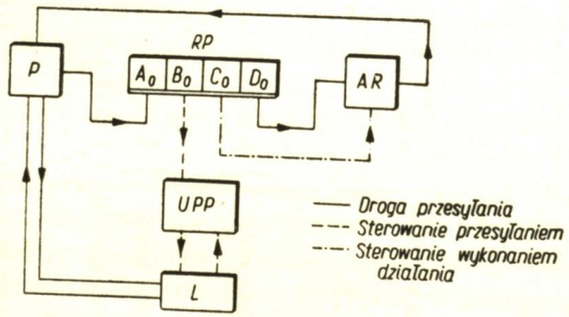
L.p.	Instrukcje	Pobieranie słowa X złożonego z cyfr: $i_0\, i_1 \ldots i_2$							
		$X = D$			$X = A$ albo $X = B$			$X = C$	
		Ostatnie pobranie		Początek ostatniego słowa [10]	Ostatnie pobranie		Początek ostatniego słowa [10]	Ostatnie pobranie	Początek ostatniego słowa [10]
I	II	III		IV	V		VI	VII	VIII
	Pierwotne:								
1	bezwarunkowe	podstawienie		$i_0 = 0$	podstawienie		$i_0 = 0$	przekształcenie	$i_0\, i_1 = 10$
2	warunkowe								$i_0\, i_1 = 13$, $i_0\, i_1 = 10$
	Wtórne:								
3	bezwarunkowe	podstawienie		$i_0 = 0$	przekształcenie		$i_0 i_1 = 10$	przekształcenie	$i_0\, i_1 = 11$
4	warunkowe								$i_0\, i_1 = 13$, $i_0\, i_1 = 10$
5	Programowana	przekształcenie	$i_0\, i_1 = 13$, $i_0\, i_1 = 10$		—		—	—	—

[10] Konieczny i dostateczny warunek spełniony przez ostatnie pobrane słowo (warunek zakończenia procesu pobierania dla danego słowa).

Tablica 2.2.2.
Zawartość słów instrukcji w chwili zakończenia pobierania

Instrukcje	$X = (i_0 i_1 \ldots\ldots i_n)$					
	$X = D$		$X = A$ albo $X = B$		$X = C$	
	$i_0\, i_1$	$i_2\, i_3 \ldots\ldots i_n$	$i_0\, i_n$	$i_2\, i_3 \ldots\ldots i_n$	$i_0\, i_1$	$i_2\, i_3 \ldots\ldots i_n$
I	II	III	IV	V	VI	VII
Pierwotne:						
bezwarunkowe	$0 i_1$	jedna z instrukcji 1-4 (łącznie z i_1)	$0 i_2$	zapis cyfrowy liczby (łącznie z i_1)	10	adres I-go rzędu
warunkowe	$0 i_1$	jedna z instrukcji 1-4 (łącznie z i_1)	$0 i_2$	zapis cyfrowy liczby (łącznie z i_1)	13 albo 10	adres instruk. albo adr. I-go rzęd.,
Wtórne:						
bezwarunkowe	$0 i_1$	jedna z instrukcji 6-8 (łącznie z i_1)	$0 i_2$	adres I-go rzędu,	11	adres II-go rzędu
warunkowe	$0 i_1$	jedna z instrukcji 6-8 (łącznie z i_1)	$0 i_2$	adres I-go rzędu,	13 albo 10	adres instruk. albo adr. I-go rzędu
Programowana	13 albo 10	adres instruk. albo adres I-go rzędu,	—	—	—	—

nia pobierania, nastąpi wykonanie w *AR* działania na znajdujących się w *RP* argumentach i przesłanie wyniku do pamięci *P*. Licznik instrukcyj,



L — licznik instukcji
P — pamięć
RP — rejestr pomocniczy
$A_0B_0C_0D_0$ — części RP służące do przechowywania słów ABCD bieżącej instrukcji
UPP — układ pobierająco-przesyłający
AR — arytmometr

Rys. 1. Uproszczony schemat blokowy maszyny cyfrowej

który w międzyczasie zwiększył swój stan o 4, będzie sterował pobieraniem następnej instrukcji.

Jeżeli w RP pojawi się słowo D z $\delta_0 = 1$ (instrukcja programowana), to dla tego słowa podobnie jak dla słów A,B,C innych instrukcji, *UPP* wykonuje czynności wyznaczone przepisem w tablicy 2.2.1. Gdy słowo D w *RP* spełni już podane w tej tablicy warunki, nastąpi przekazanie zawartości licznika instrukcyj do pierwszej instrukcji wywoływanej sekwencji, a dalej przesłanie odpowiedniej części słowa D (mianowicie $\delta_2 \ldots \delta_k$) do licznika instrukcyj (równoważne przejściu do nowej sekwencji instrukcji z przesłaniem adresu powrotu).

2.4. PRZYKŁAD POBIERANIA

Aby wyjaśnić bliżej przebieg pobierania, rozpatrzymy konkretny przykład. Maszyna ma dodać liczbę b do liczby y_n w warunkach następujących:

(1) liczba b jest dana,
(2) liczba y_n jest zapisana w pamięci pod danym adresem Y_n,
(3) sumę $b + y_n$ należy przesłać do pamięci pod adres q,
(4) adres q znajduje się w pamięci w słowie bezpośrednio następnym po słowie o adresie p,
(5) adres p jest zapisany w pamięci pod danych adresem N.

Dla rozwiązania tego zadania układamy instrukcję posiadającą, jak wiemy, w zapisie cyfrowym postać:

$$\underbrace{\alpha_0 \ldots \alpha_n}_{A} \quad \underbrace{\beta_0 \ldots \beta_n}_{B} \quad \underbrace{\gamma_0 \ldots \gamma_n}_{C} \quad \underbrace{\delta_0 \ldots \delta_n}_{D}$$

W instrukcji tej kształtujemy poszczególne słowa A,B,C i D w taki sposób, że:

(1) w słowie D; $\delta_0 = 0$ zaś $\delta_1 \ldots\ldots \delta_n$ jest zapisem cyfrowym „dodaj ..... prześlij .....” vid. tab. 2.1.1. lp. 1).
(2) w słowie A; $\alpha_0 = 0$ zaś „$\alpha_1 \ldots \alpha_n$” jest liczbą b zapisaną w obowiązującym systemie cyfrowym (np. binarnym czy dziesiętnym),
(3) w słowie B; $\beta_0\, \beta_1 = 10$ zaś „$\beta_2 \ldots \beta_k$” jest zapisem cyfrowym adresu „Y_n”. „$\beta_{k+1} \ldots \beta_n$” jest zapisem cyfrowym odstępu adresowego równego 0,
(4) w słowie C; $\gamma_0\, \gamma_1 = 12$, zaś „$\gamma_2 \ldots \gamma_k$” jest zapisem cyfrowym adresu „N”; „$\gamma_{k+1} \ldots \gamma_n$” jest zapisem cyfrowym odstępu adresowego równego 1.

Postać instrukcji łącznie z objaśnieniami znaczenia poszczególnych cyfr w słowach tej instrukcji w sposób schematyczny pokazuje ponadto tablica 2.4.1. Pobieranie słów wchodzących w skład instrukcji oraz ich przekształcanie widać z tablicy 2.4.2. Tablica ta ułożona jest w ten sposób, aby uwidocznić kolejność pobierania z pamięci słów instrukcji. Wi-

Tablica 2.4.1
Instrukcja dla przykładu na str. 87

Słowo: A	Słowo: B	Słowo: C	Słowo: D
$\underbrace{0}\ \underbrace{a_1 \ldots\ldots a_n}$	$\underbrace{10}\ \underbrace{\beta_2 \ldots \beta_k}\ \underbrace{\beta_{k+1} \ldots \beta_n}$	$\underbrace{12}\ \underbrace{\gamma_2 \ldots \gamma_k}\ \underbrace{\gamma_{k+1} \ldots \gamma}$	$\underbrace{0}\ \underbrace{\delta_1 \ldots \delta_n}$
↑ liczba b	↑ adres Y_n (adres I-go rzędu), odstęp adresowy równy 0,	↑ adres N (adres III-go rzędu) odstęp adresowy równy 1,	↑ zapis „dodaj ... prześlij ...“
sygnalizuje, że „$a_1 \ldots a_n$“ stanowi liczbę	sygnalizuje, że „$\beta_2 \ldots \beta_k$“ stanowi adres I-go rzędu.	sygnalizuje, że $\gamma_2 \ldots \gamma_k$ stanowi adres III-go rzędu	sygnalizuje, że “$\delta_1 \ldots \delta_n$“ oznacza jedną z instrukcji nr 1-9

dać z niej również, że po wykonaniu jednego pobrania w słowie B znajdzie się y_n, zaś po dwóch pobraniach w słowie C znajdzie się adres q. W tablicy 2.4.3 pokazano, w jakiej kolejności i przez jakie części maszyny są wykonywane czynności wchodzące w skład instrukcji.

Tablica 2.4.2.
Postać poszczególnych słów instrukcji w rejestrze pomocniczym na początku kolejnego pobierania

Pobieranie nr	Słowo A $\overbrace{a_0 a_1 a_2 \ldots a_n}$	Słowo B $\overbrace{\beta_0 \beta_1 \beta_2 \ldots\ldots \beta_k \beta_{k+1} \ldots \beta_n}$	Słowo C $\overbrace{\gamma_0 \gamma_1 \gamma_2 \ldots\ldots\ldots \gamma_k \gamma_{k+1} \ldots \delta_n}$	Słowo D $\overbrace{\delta_0\ \delta_1 \ldots\ldots\ldots \delta_n}$
I	II	III	IV	V
1	—	—	—	$0 \leftarrow + \Rightarrow \rightarrow$
2	—		$12 \leftarrow N \rightarrow \leftarrow 1 \rightarrow$	—— „ ——
3	—		$12 \leftarrow p+1 \rightarrow \leftarrow 0 \rightarrow$	—— „ ——
4	—		$10 \leftarrow q \rightarrow \leftarrow 0 \rightarrow$	—— „ ——
5	—	$10 \leftarrow Y_n \rightarrow \leftarrow 0 \rightarrow$	—— „ ——	—— „ ——
6	—	$0 \leftarrow Y_n \longrightarrow$	—— „ ——	—— „ ——
7	$0 \leftarrow b \rightarrow$	$0 \leftarrow Y_n \longrightarrow$	$10 \leftarrow q \rightarrow \leftarrow 0 \rightarrow$	$0 \leftarrow + \Rightarrow \rightarrow$

3. PROGRAMOWANIE

3.1. UKŁADANIE PROGRAMÓW Z POJEDYŃCZYCH INSTRUKCYJ

Omówiliśmy dotąd budowę słów, budowę instrukcji złożonych z tych słów oraz proces pobierania. Obecnie zajmiemy się zagadnieniem układa-

Tablica 2.4.3
Przebieg wykonywania instrukcji opisującej przykład na str. 87

Przedział czasu	Licznik instrukcyj L	Układ pobierająco-przesyłający UPP	Pamięć P	Rejestr pomocniczy RP	Arytmometr AR
I	II	III	IV	V	VI
1	—	—	Instrukcja znajduje się w pamięci.	—	—
2	Licznik instrukcji steruje pobraniem słowa D z pamięci	UPP analizuje wartość δ_0 w słowie D ($\delta_0 = 0$)	Pamięć dostarcza słowo D bieżącej instrukcji	Słowo D wchodzi do części D_0 rejestru RP.	—
3	L pobiera słowo C z pamięci i przesyła do rejestru RP	UPP analizuje wartość słowa D w RP (instrukcja pierwotna) oraz wartości γ_0 i γ_1 w słowie C ($\gamma_0 \gamma_1$ = 12-adres III-go rzędu) i dokonuje pobrania słowa o adresie N.	Pamięć dostarcza słowo C bieżącej instrukcji, później słowo o adresie N.	Słowo C wchodzi do części C_0 rejestru RP, później do części C_0 wchodzi słowo o adresie N.	—
4	—	UPP analizuje słowo D i słowo C w RP i dokonuje pobrania słowa o adresie $p+1$	Pamięć dostarcza słowo o adresie $p+1$	Do części C_0 wchodzi słowo o adresie $p+1$	—

Tablica 2.4.3 (c.d.)

I	II	III	IV	V	VI
5	*L* pobiera słowo *B* z pamięci i przesyła do *RP*	*UPP* analizuje słowa *D* i *B* (instrukcja pierwotna, $\beta_0\beta_1 = 10$) i pobiera słowo o adresie Y_n	Pamięć dostarcza słowo β bieżącej instrukcji, później słowo o adresie Y_n	Słowo *B* wchodzi do części B_0 rejestru *RP*, później do części B_0 wchodzi słowo o adresie Y_n	—
6	*L* pobiera słowo *A* z pamięci i przesyła do *RP*	—	Pamięć dostarcza słowo *A* bieżącej instrukcji	Słowo *A* wchodzi do części A_0 rejestru *RP*	—
		UPP steruje przesyłaniem wyniku z arytmometru do pamięci	Pod adres *q* wchodzi suma $b + y_n$	Rejestr *RP* steruje układem *UPP* i arytmometrem *AR*	Arytmometr *AR* wykonuje działanie wyznaczone słowem *D* w *RP* na argumentach zawartych w A_0 i B_0 rejestru *RP*.

nia programów z poszczególnych instrukcyj, opisanego w poprzednich paragrafach kodu instrukcyjnego.

Wprowadzamy następujące oznaczenia:

a' — adres I rzędu
a'' — adres II rzędu
a''' — adres III rzędu
$\overline{a}$ — adres instrukcji
$\underline{a}$ — adres względny instrukcji[11])

$\left.\begin{matrix}\overline{a}\ A\\ \overline{a}\ B\\ \overline{a}\ C\\ \overline{a}\ D\end{matrix}\right\}$ adresy poszczególnych słów *A*, *B*, *C*, *D* instrukcji o adresie $\overline{a}$

$\left.\begin{matrix}\underline{a}\ A\\ \underline{a}\ B\\ \underline{a}\ C\\ \underline{a}\ D\end{matrix}\right\}$ adresy poszczególnych słów *A*, *B*, *C*, *D* instrukcji o adresie $\underline{a}$

( ) — oznacza w programie słowo *A*, *B*, *C*, ew. *D* instrukcji, które będzie ulega zmianie w trakcie wykonywania tego programu

[*a*] — zawartość miejsca o adresie *a*

$A\ o_j\ B \Rightarrow C$ — przesłać liczbę będącą wynikiem działania o_j na liczbach wyznaczonych przez słowa *A* i *B* do słowa o adresie wyznaczonym przez słowo *C*[12]. ($C_j = 1, 2, 3$ ....).

$A\ o_j^*\ B \rightarrow C$ — przesłać adres będący wynikiem działania o_j na adresach wyznaczonych przez słowa *A* i *B* do słowa o adresie wyznaczonym przez słowo *C*).

Posługiwanie się kodem instrukcyjnym (tab. 2.1.1.) przy układaniu prostych programów wyjaśnimy na przykładzie programu wyciągania pierwiastka kwadratowego metodą iteracyjną wg znanego wzoru:

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right)$$

Dla ustalenia uwagi niech $y_o = 1/2$. Program podany jest w tablicy 3.1.1.

[11] Wyrażenie „adres względny" vid. Terminologia.
[12] Vid. także [37] str. 11 i 41.

Tablica 3.1.1.
Program obliczania pierwiastka kwadratowego

adres instrukcji	instrukcja	U w a g i
I	II	III
$\overline{m}$	$0 \quad + \frac{1}{2} \quad \Rightarrow Y'_n$	podstawienie $y_0 = \frac{1}{2}$
$\overline{m+1}$	$X' \quad : Y'_n \quad \Rightarrow 1\,A$	obliczenie y_{n+1}
$\overline{m+2}$	$( ) \quad + Y'_n \quad \Rightarrow 1\,A$	
$\overline{m+3}$	$( ) \quad \cdot \frac{1}{2} \quad \Rightarrow Y'_{n+1}$	
$\overline{m+4}$	$Y'_{n+1} - Y'_n \quad \Rightarrow G'$	stwierdzenie czy $g^2 = (y_{n+1} - y_n)^2$ jest równe zeru [13]
$\overline{m+5}$	$0 \quad + Y'_{n+1} \Rightarrow Y'_n$	
$\overline{m+6}$	$G' \quad \cdot G' \quad \Rightarrow 1\,B$	
$\overline{m+7}$	$0 \quad ; ( ) \quad \stackrel{?}{\Rightarrow} -6$	
$\overline{m+8}$	$Z'''o_i^* \; \dot{Z}''' \rightarrow \dot{Z}'''$	instrukcja „Stop" [14]

Program składa się niemal wyłącznie z instrukcji pierwotnych (instrukcje o adresach $\overline{m}$ do $\overline{m+7}$), w tym tylko jedna warunkowa (o adresie $\overline{m+7}$), oraz z jednej instrukcji wtórnej (o adresie $\overline{m+8}$). W pierwszej kolumnie tablicy 3.1.1. znajduje się adres instrukcji zapisanej w tym samym wierszu kolumny drugiej. W trzeciej kolumnie tablicy jest wyjaśniony efekt działania kolejnych instrukcji programu. Dla zwiększenia czytelności programu adresy liczb oznaczonych małymi literami zostały oznaczone odpowiednimi literami wielkimi, np. adresem liczby x jest X.

Dla wyjaśnienia przebiegu pobierania oraz dla podkreślenia różnic w stosunku do znanych autorowi kodów instrukcyjnych[15] rozpatrzymy przykładowo niektóre instrukcje podanego programu:

1) Jak widać z tablicy 3.1.1., w słowach A i B stanowiących część instrukcji o adresie "m" zapisane są odpowiednio liczby 0 i $^1/_2$. Przy wykonywaniu tej instrukcji nastąpi dodanie do siebie tych liczb i przesłanie wyników pod adres Y_n. Dla każdego słowa wymienionej instrukcji układ pobierający wykonuje tylko jedno pobranie (tab. 2.2.1. lp. 1).

2) W instrukcji o adresie "$m+1$" dla słów A i B układ pobierający wykonuje po dwa pobrania. Pobierze mianowicie najpierw słowa A i B, wchodzące w skład instrukcji i zawierające odpowiednio adresy I rzędu X' i Y' a następnie wg tych adresów pobierze liczby x i y_n. Wynik podzielenia x przez y_n zostanie przesłany pod adres względny $\underline{1}$ A. Zapis ilorazu $\frac{x}{y}$ wystąpi więc jako słowo A instrukcji $\overline{m+2}$.

[13] Należy pamiętać, że w maszynie ilość cyfr w słowie jest stała (i oczywiście niezmienna).
[14] Z jest adresem miejsca w pamięci zawierającego własny adres III rzędu.
[15] Vid. np. [11], [27], [28], [34], [36], [54].

3) W instrukcji "$\overline{m+2}$" dwa pobrania wystąpią tylko dla słowa B. W słowie A tej instrukcji będzie się znajdował odpowiedni argument przesłany tam jako wynik działania instrukcji "$\overline{m+1}$".

4) Instrukcja "$\overline{m+7}$" jest instrukcją pierwotną, warunkową. W słowach A i B tej instrukcji liczby 0 i g^2 znajdą się już po pierwszym pobraniu. Jeżeli różnica tych liczb będzie mniejsza od zera, tzn. jeżeli $g^2 = (y_{n+1} - y_n)^2 \neq 0$, to następną instrukcją będzie instrukcja o adresie względnym $\underline{-6}$, czyli instrukcja "$\overline{m+1}$".

5) Instrukcja "$\overline{m+8}$" jest równoważna instrukcji „Stop" ze względu na to, że zawartość adresu Z''' jest równa Z''' (wystąpi nie kończące się pobieranie Z''' — tab. 2.2.1. lp. 3).

Zastępowanie adresów względnych, występujących w programie 3.1.1., odpowiednimi adresami zwykłymi zostanie objaśnione niżej.

Omówiliśmy przykład programu ułożonego za pomocą wprowadzonego kodu instrukcyjnego. W następnych programach zajmiemy się następującymi zagadnieniami programowania:

1) Zależność adresów, wchodzących w skład niektórych instrukcji programu, od położenia w pamięci programu lub jego części składowych (np. w tablicy 3.1.1. instrukcje $\overline{m+1}$, $\overline{m+2}$, $\overline{m+6}$, $\overline{m+7}$).

2) Zamiana adresów (adres adresu) równoważna podstawieniu adresu[16].

3) Wykonywanie tych samych instrukcji na różnych argumentach (rozwiązywanie numeryczne równań różniczkowych cząstkowych metodą siatek, obliczenia na wektorach, macierzach itp.).

3.2. ZALEŻNOŚĆ PROGRAMU OD POŁOŻENIA W PAMIĘCI

Zależność tę można usunąć dodając (podczas wprowadzania programu do pamięci) do instrukcji zawierających adres względny, rzeczywisty adres instrukcji, w odniesieniu do której były liczone adresy względne[17]. Przy wprowadzonej metodzie adresowania i pobierania istnieje dodatkowo możliwość wykorzystania zawartości licznika instrukcyj.

Niech Ł będzie adresem adresu *pierwszego* słowa bieżącej instrukcji (odpowiednia część tego słowa jest utworzona przez zawartość licznika instrukcyj — w technicznej realizacji nie nastręcza to szczególnych trudności), zaś L adresem adresu *bieżącej instrukcji* (odpowiednia część tego

[16] Vid. [2] str. 187 — "substitution et exploration itérées" i [12] tablica II instrukcje 13 i 19.
[17] Vid. także [50], [51] str. 19.

słowa jest również utworzona przez zawartość licznika instrukcyj). Instrukcje $\overline{m+1}$ i $\overline{m+7}$ z tablicy 3.1.1. miałyby wówczas postać:

$$X' : Y'_n \Rightarrow Ł' 1 ; \quad 0 ; ( ) \stackrel{?}{\Rightarrow} L''' - 6$$ [18]

gdzie na "1" i „—6" wykorzystuje się cyfry $\sigma_{k+1} \ldots \sigma_n$ a na Ł i L cyfry $\sigma_2 \ldots \sigma_k$ odpowiedniego słowa.

Układanie programu w którym później wystąpi:

1) zmiana położenia instrukcyj

2) wstawianie dodatkowych czy też wykreślanie zbędnych instrukcyj, co mimo zastosowania adresów względnych powodowałoby konieczność zmiany niektórych adresów, zostało ułatwione przez zastosowanie zmiennego systemu adresów[19]. Zmienny układ adresów polega na tworzeniu w odpowiedniej chwili skorowidza[20] adresów tych słów, które będą później przedmiotem innych instrukcji i magazynowaniu tego skorowidza w pamięci. W instrukcjach nie używa się wówczas adresu słowa (który nie jest jeszcze znany), lecz adresu miejsca w skorowidzu, dokąd adres tego słowa zostanie przesłany np. podczas wprowadzania programu do pamięci. Przyjęta przez nas metoda adresowania pozwala na proste korzystanie ze skorowidza adresów tworzonego w tym systemie.

Tablica 3.2.1.
Program obliczania pierwiastka kwadratowego
(przy wykorzystaniu skorowidza adresów)

Adres instrukcji	Instrukcje zewnętrzne	Instrukcje	Uwagi
I	II	III	IV
$\overline{m}$		$0 + \frac{1}{2} \Rightarrow Y'_n$	przesłanie $\frac{x}{Y_n}$ pod adres $\overline{(m+2)A}$
$\overline{m+1}$	$N4A$	$X' : Y'_n \Rightarrow N'''1$	
$\overline{m+2}$	$N1A$	$( ) + Y'_n \Rightarrow N'''2$	
$\overline{m+3}$	$N2A$	$( ) \cdot \frac{1}{2} \Rightarrow Y'_{n+1}$	
$\overline{m+4}$		$Y'_{n+1} - Y'_n \Rightarrow G$	
$\overline{m+5}$		$0 + Y'_{n+1} \Rightarrow Y'_n$	
$\overline{m+6}$		$G' \cdot G' \Rightarrow N'''3$	
$\overline{m+7}$	$N3B$	$0 ; ( ) \stackrel{?}{\Rightarrow} N'''4$	
$\overline{m+8}$		$Z'''\sigma_j^* Z''' \Rightarrow Z'''$	

[18] Jak widać z tab. 2.2.1. można zastąpić ttuaj Ł' przez Ł", albo Ł''' albo Ł, zaś L''' przez L.

[19] "Floating adress system", [52], [25].

[20] Tworzenie tego skorowidza ("directory") odbywa się za pomocą odpowiednich "control combination" w [52], zaś "instruction immèdiates" w [2].

Zakodowany przy użyciu zmiennego układu adresów program 3.1.1. podajemy w tablicy 3.2.1.

Zapisy postaci "NnQ" (gdzie $Q = A, B, C, D$, zaś $n = 1, 2, \ldots$) znajdujące się w drugiej kolumnie tablicy 3.2.1 zwane „instrukcjami zewnętrznymi" sygnalizują, że adres słowa Q instrukcji stojącej w tym samym wierszu w kolumnie trzeciej należy umieścić w n-tym miejscu skorowidza, którego adres (tzn. adres jego pierwszej pozycji) znajduje się pod adresem N. Instrukcje zewnętrzne stojące w kolumnie drugiej są wykonywane podczas wprowadzania programu do pamięci.

3.3. ZAMIANA ADRESÓW

Realizujemy ją przez użycie adresu słowa (oczywiście adresu odpowiedniego rzędu) zawierającego adres, który chcemy podstawić[21].

Po wykonaniu jednego pobrania (zgodnie z tablicą 2.2.1) nastąpi zamiana adresu.

Jak wiadomo, zastąpienia adresu w instrukcji można dokonywać także za pomocą specjalnych instrukcyj[22].

3.4. WYKONYWANIE TYCH SAMYCH INSTRUKCJI NA RÓŻNYCH ARGUMENTACH

Dla argumentów umieszczonych kolejno w pamięci rozwiązuje się to zadanie (celem uproszczenia i skrócenia programu) za pomocą rejestrów, których zawartość jest dodawana do instrukcji *przed jej wykonaniem*, powodując zmianę odpowiedniego adresu[23].

W proponowanym systemie adresowania jest to także możliwe, gdyż $\sigma_2 \ldots \sigma_k$ jest po prostu adresem miejsca [24] w pamięci, którego zawartość należy dodać (pod pewnymi warunkami podanymi w tablicy 2.2.1) do instrukcji przed jej wykonaniem.

Jako przykład wielokrotnego wykonywania tych samych instrukcji na różnych argumentach podajemy program skalarnego mnożenia wektorów (tablica 3.4.1). Program opisuje obliczenie iloczynu skalarnego.

$$g = \sum_{i=1}^{n} a_i b_i$$

Jak łatwo można stwierdzić analizując program 3.4.1, zwiększenie wartości wskaźnika i o 1 odpowiada przejściu do nowej pary liczb (o adresach o 1 większych), gdyż liczby a_i i b_i są umieszczone odpowiednio w pamięci.

[21] Cyfry $\sigma_{k+1} \ldots \sigma_n$ winny tworzyć wtedy liczbę równą zeru.

[22] Vid. instrukcje 18 i 19 tablicy II podanej w [12].

[23] [38] — "i-register", [22] — "B-tube", [39] — "Adressenaddenderigster"..

[24] Miejsce to może być dowolnym miejscem pamięci, gdyż ilość cyfr $\sigma_2 \ldots \sigma_k$ jest tak dobrana, że pozwala na penumerowanie wszystkich miejsc pamięci. Stąd wniosek, że własności "i — rejestru" są rozszerzone na całą pamięć.

Tablica 3.4.1
Program mnożenia skalarnego wektorów

Instrukcje	Uwagi
$0' \quad +1' \quad \to I'$	podstawienie $i = 1$
$0 \quad -S' \quad \Rightarrow S'$	wyzerowanie akumulatora[25]
$I' \quad \cdot I''b \Rightarrow S'$	obliczenie oraz dodanie do sumy częściowej[26]
$I'' \quad +1' \quad \to I''$	podstawienie i + 1na miejsce i
$I'' \quad ; n' \quad \overset{?}{\to} Ł''' - 2$	sprawdzenie czy $i \leqslant n$
$0 \quad +S' \quad \Rightarrow G'$	przesłanie wyniku
$Z'''o_i^* \quad Z''' \to Z'''$	instrukcja "S t o p"

Zalety wprowadzonej budowy instrukcji, umożliwiającej podstawienie argumentu bezpośrednio do instrukcji, pokażemy jeszcze na przykładzie programu mnożenia macierzy przez wektor (tablica 3.4.2).

Tablica 3.4.2 cz. I
Rozmieszczenie wyrazów wektora W

Adres pierwszego wyrazu wektora W	Składniki wektora W	Adres ostatniego wyrazu wektora W
I	II	III
$w+1$	$w_1 w_2 \ldots w_j \ldots w_{n-1}, w_n$	$w+n$

Tablica 3.4.2 cz. II
Rozmieszczenie macierzy A i adresów wektora W w pamięci

Adres pierwszego wyrazu wiersza	Wyrazy macierzy	Adres kolejnych wyrazów wektora	Adres ostatniego wyrazu wiersza
$a+1$ $a+n+2$	$a_{11}, \quad a_{11} \ldots\ldots a_{ij} \ldots\ldots a_{1,n-1} \ldots a_{1n}$ $a_{21}, \quad a_{22} \ldots\ldots a_{2j} \ldots\ldots a_{2,n-1} \ldots a_{2n}$	c_1' c_2'	$c+1$ $c+n+2$
$a+(i-1)(n+)+1$	$a_{i1} \quad a_{i2} \ldots\ldots a_{ij} \ldots\ldots a_{i,n-1}, \quad a_{in}$	c_i'	$c+(i+1)(n+2)+1$
$a+n^2-n-1$ $a+n^2$	$a_{n-1,1}\, a_{n-1,2} \ldots a_{n-1,j} \ldots a_{n-1,n-1}, a_{n-1,n}$ $a_{n1} \quad a_{n2} \ldots\ldots a_{nj} \ldots\ldots a_{n,n-1}, \quad a_{nn}$	c_{n-1}' c_n'	$c+n^2-n-1$ $c+n^2$

[25] S jest adresem miejsca w pamięci posiadającego własność sumowania — akumulator.
[26] Gdzie a jest adresem a_0, zaś b adresem b_0.

Część I tablicy 3.4.2 pokazuje rozmieszczenie w pamięci wyrazów wektora W. W kolumnie pierwszej znajduje się adres pierwszego wyrazu wektora W tzn. adres w_1. W kolumnie trzeciej znajduje sią adres w_n. W części II tablicy 3.4.2 kolejne wiersze kolumny drugiej zawierają kolejne wiersze macierzy A. Kolumna trzecia zawiera adresy, pod którymi mają się znaleźć wyrazy wektora C będącego wynikiem iloczynu macierzy A przez wektor W. Pierwsza kolumna zawiera adresy pierwszych, a czwar-

Tablica 3.4.3
Program mnożenia macierzy przez wektor

Instrukcje zewnętrzne	Instrukcje	Uwagi
I	II	III
	$\lvert w+1\rvert' \div \lvert w+n\rvert' \to N'''1$	instrukcja powodująca przesłanie składników wektora W do słów B instrukcyj od N 1 do N 4[27]
$N3A$	$0' \quad +1' \quad \to I''$ $I''a \quad \div I''c \quad \to N'''2$	przesłanie kolejnego wiersza macierzy, łącznie z adresem, pod którym ma się znaleźć obliczony składnik wektora
$N2A, N1B$	$(a_{i1}) \cdot (w_1) \to S'$ $(a_{i2}) \cdot (w_2) \Rightarrow S'$ ..... $(a_{ij}) \cdot (w_j) \Rightarrow S'$	
$N4B$	$(a_{in-1}) \cdot (w_{n-1}) \Rightarrow S'$ $(a_{in}) \cdot (w_n) \Rightarrow S'$ $(c_i') \quad +0' \quad \to L'''0$	instrukcja równoważna instrukcji "nic nie rób"
	$0 \quad +S' \quad \to L'''-1$	przesłanie $\sum_{j=1}^{n} a_{ij} \cdot w_j$ pod adres $c_i <$
	$0 \quad -S' \quad \Rightarrow S'$ $I'' \quad +\lvert n+1\rvert' \to I''$	wyzerowanie akumulatora przejście do następnego wiersza macierzy
	$I'' \quad ; \lvert n^2+1\rvert' \overset{?}{\to} N'''3$	sprawdzenie czy zostały wykonane wszystkie mnożenia wektora przez wiersze macierzy
	$Z'''o_j^* \quad Z''' \to Z'''$	"S t o p"

[27] Realizacja takiego przesyłania byłaby bardzo prosta w maszynie z pamięcią typu opóźnieniowego, jeżeli się dodatkowo przyjmie, że poszczególne słowa instrukcji znajdują się w sąsiednich rurach (pamięć rtęciowa) lub ścieżkach (pamięć bębnowa). Pobranie instrukcji odbywałoby się wówczas przez pobieranie oddzielnych słów względnie przez jednoczesne pobieranie z czterech rur albo ścieżek.

ta ostatnich wyrazów wierszy utworzonych przez wiersze kolumny drugiej i trzeciej. Po tych wstępnych informacjach możemy już podać program mnożenia macierzy przez wektor (tablica 3.4.3).

Program podany w tablicy 3.4.3 nie wymaga dodatkowych objaśnień. Na podkreślenie zasługuje tylko działanie instrukcyj przesyłania zespołowego, podstawiających od razu całe zespoły liczb na właściwe miejsca do zespołu instrukcyj.

Wprowadzona w niniejszej pracy budowa instrukcyj jest szczególnie korzystna przy stosowaniu metody tzw. liniowania. Metoda ta polega na zastąpieniu pętli indukcyjnych, zwłaszcza wewnętrznych, przez wypisaną explicite „linię" instrukcyj operacyjnych, które byłyby wykonane przy wielokrotnym przejściu pętli instrukcji. Poza tym, dla pewnych problemów unika się wykonywania zbędnych instrukcji operacyjnych [28]. Oczywiście metoda ta prowadzi do znacznego zwiększenia ilości instrukcyj w programie. Można tu jednak z powodzeniem wykorzystać dodatkową pamięć na taśmie magnetycznej[29].

Zastosowanie wprowadzonego tu kodu instrukcyjnego do metody liniowania pokażemy na przykładzie wyliniowanego programu rozwiązania układu równań liniowych metodą Gauss—Seidela. Obliczenie odbywa się według zależności:

$$x_i^{|k|} = \frac{1}{a_{ii}}\left[C_i - \left(\sum_{j=1}^{i-1} a_{ij} \cdot x_j^{|k|} + \sum_{j=i+1}^{n} a_{ij} \cdot x_j^{|k-1|}\right)\right]$$

gdzie $x_i^{|k|}$ jest wartością i — tej niewiadomej obliczoną w k-tej iteracji (tablica 3.4.4).

Program podany w tablicy 3.4.4 pokazuje obliczenie tylko wartości zmiennej x_i. Pełny program składa się z n takich programów (dla $i = 1, 2 \ldots n$) oraz z instrukcji opisujących warunek zakończenia iteracji. Cały program jest powtarzany wielokrotnie, aż warunek zakończenia iteracji zostanie spełniony. Przy układaniu programu nie wypisuje się instrukcji, w których $a_{ij} = 0$ eliminując w ten sposób zbędne instrukcje pierwotne. Przed zaczęciem liczenia pod adresami $X'_1\ X'_2 \ldots X'_i \ldots X'_n$ znajdują się pierwsze przybliżenia niewiadomych.

Wypisanie ręczne pełnego programu dla dużych wartości n byłoby bardzo pracochłonne, np. dla pełnego układu równań, w którym $n = 100$ należałoby wypisać ponad 10 tys. instrukcji. Aby tego uniknąć, można zastosować specjalny program, układający z programu „pętlowego" (w którym występuje podstawienie i liczenie adresów) program liniowy,

[28] Np. przy mnożeniu macierzy przez wektor nie mnoży się przez składniki równe zeru, vid. także [37] str. 34.
[29] Vid. [18] str. 1253.

Tablica 3.4.4
Program rozwiązania układu równań liniowych

Instrukcje	Uwagi
I	II
$0 \quad -S' \quad \Rightarrow S'$	wyzerowanie akumulatora
$a_{i1} \cdot X'_1 \Rightarrow S'$	
$a_{i2} \cdot X'_2 \Rightarrow S'$	
.....	
$a_{i\,i-1} \cdot X'_{i-1} \Rightarrow S'$	
$a_{i,i+1} \cdot X'_{i+1} \Rightarrow S'$	Obliczenie x
.....	
$a_{in} \cdot X'_n \Rightarrow S'$	
$0 \quad -C_i \quad \Rightarrow S'$	
$S' \quad : a_{ii} \Rightarrow g'$	
$0 \quad -g' \quad \Rightarrow X'_i$	
$0 \quad -S' \quad \Rightarrow S'$	wyzerowanie akumulatora
$a_{i+1,1} \cdot X'_1 \Rightarrow S'$	Obliczenie x_{i+1}
$a_{i+1,2} \cdot X'_2 \Rightarrow S'$	
.....	

wstawiający w odpowiednie instrukcje zamiast adresów liczby, oraz eliminujący instrukcje zawierające $a_{ij} = 0$[30].

3.5. PROGRAMY BIBLIOTECZNE I SKŁADANIE PROGRAMÓW

Biblioteka programów[31] uwalnia programującego od konieczności każdorazowego ułożenia i wypisania wszystkich instrukcji danego programu oraz zmniejsza znacznie możliwość powstawania błędów. Włączanie programów pomocniczych[32], ułożonych specjalnie dla danego programu, jest ułatwione, jeżeli się zastosuje skorowidz programów. Metoda tworzenia i korzystania ze skorowidza programów będzie identyczna z metodą tworzenia skorowidza adresów, opisaną w 3.2.

Dla podstawowych programów bibliotecznych można przewidzieć w skorowidzu stałe miejsca do przechowywania ich adresów. Dla rzadziej używanych programów bibliotecznych i programów pomocniczych zarezerwować można tylko pewną ilość miejsc[33].

Skorowidz programów w zasadzie może się znajdować w dowolnym miejscu pamięci z tym, że jego adres (tzn. adres pierwszej pozycji skorowidza) musi być umieszczony w z góry określonym miejscu pamięci. Adres tego miejsca będziemy oznaczali przez "F" a przez „Fh" adres miejsca w skorowidzu zawierającego adres h-go programu.

[30] Vid także [36] str 34 i [18] str. 1253.
[31] Vid [50], [19], [28], [29].
[32] "auxiliary subroutines".
[33] Vid także [19] str. 31, [3] str. 44.

Oczywiście, przy każdorazowym wprowadzaniu do pamięci podstawowych programów bibliotecznych (wchodzących w skład programu liczonego problemu), do stałych miejsc w skorowidzu są wprowadzane aktualne adresy pierwszych instrukcji tych programów. Aktualne adresy programów pomocniczych są przesyłane do każdorazowo wybieranych miejsc, zarezerwowanych w tym celu w skorowidzu [34].

Tablica 3.5.1 podaje program główny[35] tablicowania funkcji

$$f(x) = \exp(-\cos ax) \cdot \sin^2 bx \quad \text{dla } x = 0\,(0{,}01)\,0{,}98$$

Tablica 3.5.2 pokazuje rozmieszczenie w pamięci programów bibliotecznych „exp", „cos", „sin", skorowidza programów i skorowidza adresów. Część a tablicy 3.5.2 zawiera skorowidz programów. W pierwszej kolumnie znajduje się litera F — adres pierwszego miejsca skorowidza. Część b tablicy pokazuje, że pierwsze instrukcje programów „exp", „cos", „sin" mają adresy E, G, S.

Pod pozycjami $F2$, $F3$, $F4$, skorowidza programów znajdują się odpowiednie adresy programów „exp", „cos" i „sin" dostarczone tam podczas wprowadzania programów do pamięci[36].

Część b tablicy podaje zawartość skorowidza adresów, zastosowanego dla oznaczenia określonych słów programu głównego.

Dwie pierwsze instrukcje programu z tab. 3.5.1. nie wymagają dodatkowych objaśnień. Instrukcja $\overline{m+2}$ powoduje obliczenie ax i przesłanie do słowa A instrukcji $\overline{m+3}$. Instrukcja $\overline{m+3}$ jest instrukcją programowaną. Zgodnie z przepisem z tablicy 2.2.1. lp. 5 wykonanie instrukcji programowej będzie miało następujący przebieg:

1) pobranie $[F]$[37] (tzn. adresu skorowidza programów) i utworzenie adresu $[F]+3$

($[F]$ jest adresem III rzędu).

2) pobranie $[[F]+3] = G$

(tzn. adresu programu „cos")

3) przesłanie $[L]$ do słowa A instrukcji G oraz $[Ł]$ do słowa B instrukcji G (tj. przesłanie adresu powrotu)[38].

4) przesłanie $G+1$ do licznika instrukcji (równoznaczne pobraniu instrukcji $G+1$ jako następnej instrukcji).

[34] Jeżeli pewien podstawowy, biblioteczny program korzysta z programu pomocniczego (np. program całkowania wywołujący program pomocniczy obliczający funkcję całkowaną), to dla takiego programu pomocniczego jest przewidziane także stałe miejsce w skorowidzu.

[35] "main program".

[36] Wykonują to odpowiednie instrukcje zewnętrzne znajdujące się na początku programów „exp", „cos" i „sin".

[37] $[F]$ oznacza zawartość słowa o adresie F.

[38] „linking data" [23] str. 342.

Tablica 3.5.1

Program główny tablicowania funkcji $f(x) = \exp(-\cos ax) \cdot \sin^2 bx$

Adresy instrukcji	Instrukcje zewnętrzne	Instrukcje	Uwagi
I	II	III	IV
$\overline{m}$		$0' + f_0' \rightarrow N'''3$	przesłanie adresu f (x_0
$\overline{m+1}$		$0 + 0 \Rightarrow X'$	podstawienie $x_0 = 0$
$\overline{m+2}$	$N4A$	$a \cdot X' \Rightarrow Ł'''1$	obliczenie ax
$\overline{m+3}$		$(); N'''0; Z'''; F'''3$	wywołanie programu obliczającego „cos"
$\overline{m+4}$	$N0B$	$0 - () \Rightarrow Ł'''1$	przesłanie — cos ax
$\overline{m+5}$		$(); N'''Z; Z'''; F'''2$	wywołanie programu obliczającego „exp"
$\overline{m+6}$		$b \cdot X' \Rightarrow Ł'''1$	obliczenie bx
$\overline{m+7}$		$(); N'''1; Z'''; F'''5$	wywołanie programu obliczającego „sin"
$\overline{m+8}$	$N1A$	$() \cdot Ł'''0 \Rightarrow Ł'''1$	obliczenie sinbx
$\overline{m+9}$	$N2A$	$() \cdot () \Rightarrow L'''1$	obliczenie $\sin^2 bx \exp(\cos ax)$
$\overline{m+10}$	$N3A$	$i' + 1' \rightarrow Ł'''0$	obliczenie adresu wartości $f(x)$ dla kolejnych wartości x
$\overline{m+11}$		$X' + 0{,}01 \Rightarrow X'$	podstawienie $x + \triangle x$ na miejsce x
$\overline{m+12}$		$X'; 0{,}99 \overset{?}{\Rightarrow} N'''4$	sprawdzenie czy $x < x_k = 0{,}99$
$\overline{m+13}$		$Z'''o_1^*Z''' \rightarrow Z'''$	„Stop"

W części C, tablicy 3.5.2 podano pierwsze instrukcje programów „exp", „cos", „sin". W słowach A i B tych instrukcji znajdują się adresy powrotu przesłane tam jako wynik działania instrukcji programowanych $\overline{m+3}$, $\overline{m+5}$, $\overline{m+7}$ programu z tablicy 3.5.1.

Adresy powrotu umożliwiają zarówno powrót do programu wywołującego, jak i pobranie z programu wywołującego parametrów dla programu wywołanego[39]. Adresy te mogą być użyte do „wyprodukowania" instrukcji pobierających parametry[40].

[39] Np. dla instrukcji programowanej $\overline{m+3}$ parametrami są słowa o adresie $(m+3)$ A (wartość argumentu) i słowo o adresie $(m+3)$ B (adres, pod który należy przesłać wynik obliczony przez program „cos").

[40] Vid. [50] str. 22.

Dostęp do parametrów można u nas zrealizować, poza tym, bez „produkowania" instrukcji pobierających parametry.

Tablica 3.5.2

a) Skorowidz programów

Adres pierwszego miejsca skorowidza	Adres względny	Zawartość poszczególnych miejsc skorowidza	Uwagi
I	II	III	IV
F	0	—	—
	1	—	—
	2	E	E — adres programu „exp"
	3	G	G — adres programu „cos"
	4	—	—
	5	S	S — adres programu „sin"
	6	—	—

b) Skorowidz adresów

Adres pierwszego miejsca skorowidza	Adres względny	Zawartość poszczególnych miejsc skorowidza	Uwagi
I	II	III	IV
N	0	$\|m+4\|'B$	adresy określonych słów programu głównego
	1	$\|m+8\|'A$	
	2	$\|m+9\|'A$	
	3	$\|m+10\|'A$	
	4	$\|m+2\|'A$	

c) Zawartość pierwszej instrukcji programów „exp", „cos", „sin"

Adres pierwszej instrukcji programu	instrukcja	Uwagi
I	II	III
E	$\|\overline{m+5}\|;\|m+5\|'$	adres powrotu
G	$\|\overline{m+3}\|;\|m+3\|'$	adres powrotu
S	$\|\overline{m+7}\|;\|m+7\|'$	adres powrotu

Niech $X\ o_j\ Y \Rightarrow W$, będzie jedną z instrukcji wywołanego programu i niech np. X wyznacza parametr, który ma być pobrany z programu wywołującego. Zakładamy, że parametr znajduje się w n-tym słowie programu wywołującego (licząc od instrukcji programowanej, która wywołała dany program). Instrukcja ta dla automatycznego pobrania parametru powinna mieć postać

$$P''_n\ o_j\ Y \Rightarrow W$$

gdzie P jest adresem pierwszej instrukcji wywołanego programu. Wobec tego

$[P]$ adres instrukcji wywołującej,

a stąd

$[[P] + n]$ adres parametru.

Z powyższego widać, że przy wprowadzaniu programu do pamięci należałoby wprowadzać do słów, pobierających parametry z wywołującego programu, adres pierwszej instrukcji programu[41].

Powrót do programu głównego może się odbywać za pomocą instrukcji programowanej (korzystając z adresu powrotu) lub za pomocą zwykłej instrukcji wtórnej przesyłającej adres powrotu zwiększony o k (k — ilość słów zajętych na parametry) do licznika instrukcyj.

Podana metoda wywoływania programów i pobierania parametrów w dużej mierze ułatwia układanie programów takich zagadnień, w których w szerokim zakresie korzysta się z biblioteki programów i programów pomocniczych.

3.6. UWAGI DODATKOWE

Jak wiadomo, w zwykłej maszynie trójadresowej wykonanie instrukcji obejmuje następujące czynności:

a) pobranie z pamięci argumentów,
b) wykonanie działania,
c) przesłanie wyniku do pamięci,
d) pobranie z pamięci następnej instrukcji.

Dwa argumenty wchodzące do działania są pobierane z pamięci w niektórych maszynach jednocześnie[42], w innych kolejno[43] i przesyłane do odpowiednich rejestrów arytmometru. Jak widać, chociażby z programu w tablicy 3.2.1, przyjęta budowa instrukcji sprawia, że dla pewnych instrukcji nie potrzeba pobierać z pamięci jednego (instrukcje $\overline{m+2}$, $\overline{m+5}$), a nawet dwóch argumentów (instrukcje $\overline{m}$, $\overline{m+3}$, $\overline{m+7}$), gdyż argumenty te stanowią część składową bieżącej instrukcji, czyli zostają pobrane z pamięci w ramach pobierania instrukcji. Używając odpowiednie słowo

[41] Np. przy pomocy metody podanej w [51], gdzie stosuje się do tego celu „code letter".

[42] Np. maszyna „MOSAIC" [4], [5].

[43] Np. maszyna „SEAC" [14].

instrukcji, w której występuje po raz pierwszy adres zmiennej, jako miejsca służącego do magazynowania wartości tej zmiennej, można jeszcze bardziej obniżyć ilość operacji pobierania z pamięci. Gdybyśmy wybrali np. na miejsca magazynowania wartości x i y_n słowa $(m+1)\,A$; $(m+1)\,B$, to instrukcja $\overline{m+1}$ miałaby postać

$$x : y_n \Rightarrow N''' 1$$

i tym samym w tej instrukcji nie występowałoby oddzielne pobieranie argumentów dzielenia. Oczywiście, należy dążyć do umieszczenia słowa tworzącego wartość zmiennej w takiej instrukcji, która w danym programie jest wykonywana największą ilość razy.

Dla maszyn z wolną pamięcią główną ilość operacji pobierania i przesyłania do pamięci ma istotny wpływ na szybkość liczenia.

W istniejących maszynach z wolną pamięcią typu opóźnieniowego znaczna część czasu wykonywania instrukcji (tzw. czas oczekiwania) jest zużyta na pobranie z pamięci instrukcji oraz pobranie i przesłanie wielkości, których ona dotyczy. W celu eliminacji czasu oczekiwania w maszynach tego typu używa się szeregu metod[44]. Jedna z nich polega na zastosowaniu buforowej pamięci[45] instrukcji, o pojemności kilku instrukcji napełnianej automatycznie w miarę ich wykonywania. Pamięć buforowa instrukcji jest także elementem składowym maszyny cyfrowej pracującej z wprowadzonym kodem instrukcyjnym. Schemat blokowy tej maszyny opisano w 4.2.

3.7. TECHNIKA INTERPRETACYJNA[46]

Technika interpretacyjna rozwinięta została w związku z potrzebą liczenia na maszynach ze stałym przecinkiem[47] i o określonej ilości cyfr w słowie, zagadnień wymagających ruchomego przecinka[48], zwiększonej dokładności[49], arytmetyki liczb zespolonych itp. Polega ona na tym, że programujący układa w dowolnym „kodzie" instrukcyjnym „program". Odpowiedni dla wybranego „kodu" specjalny program, tzw. program interpretacyjny, dokonuje analizy kolejnych „instrukcji" ułożonego „programu" i wywołuje z tzw. bloku programów funkcyj[50] odpowiadający danej „instrukcji" program.

[44] [21], [31], [54], [1].
[45] „buffer storage".
[46] „interpretive techniques" vid [12], [24], [25], [27], [28], [29], [30].
[47] „fixed point".
[48] „floating point".
[49] „multiple precision".
[50] „arithematical and organisational function block" — zespół bibliotecznych programów funkcyj podstawowych dla danego „kodu".

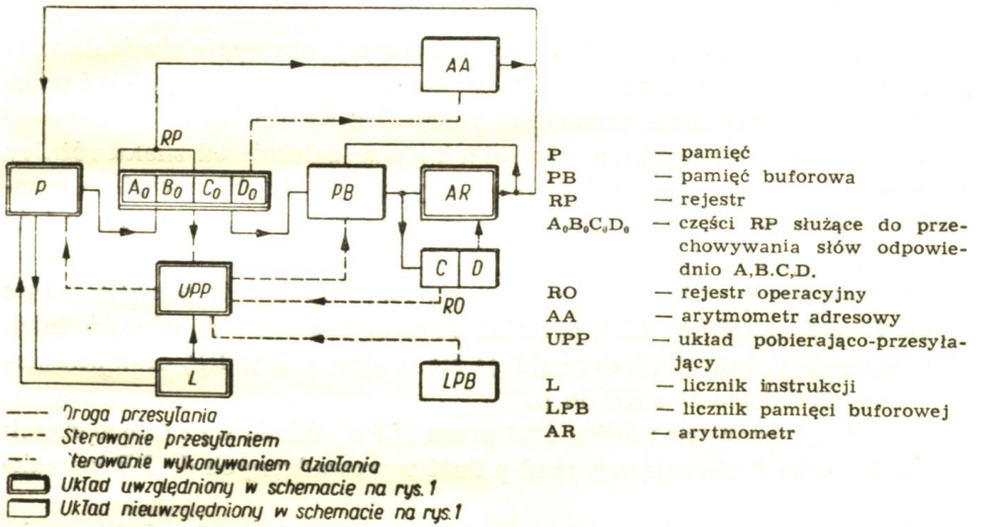
Technika ta dla pewnych zagadnień posiada znaczną przewagę nad zwykłą metodą programowania, gdyż umożliwia programującemu wybór dowolnego „kodu" dla poszczególnych problemów oraz uwalnia od wypisywania w programie głównym instrukcji związanych z wywoływaniem programów z bloku funkcyj.

Wprowadzenie instrukcji programowanej łącznie z wybranym systemem adresowania, której działanie omówiono w 3.5, umożliwia także wybór dowolnego „kodu" i uwalnia od układania specjalnych instrukcji przesyłających adresy powrotu i argumenty (liczby i adresy będące przedmiotem działania wywołanego programu). Warto przy tym dodać, że unika się wprowadzanego przez program interpretacyjny zwiększenia czasu liczenia problemu.

4. BUDOWA MASZYNY

4.1. OGÓLNY SCHEMAT MASZYNY

W 2.3 opisaliśmy w sposób uproszczony schemat maszyny cyfrowej pracującej według wprowadzonego tu kodu instrukcyjnego. Obecnie opi-



Rys. 2. Schemat blokowy maszyny cyfrowej

szemy tę maszynę nieco szczegółowiej, podkreślając przy tym możliwości zwiększenia szybkości liczenia.

Jak już pisaliśmy w 2.3, licznik instrukcyj (vid. rys. 2) steruje poprzez układ pobierająco — przesyłający pobieraniem kolejnych instrukcji z pamięci i przesyłaniem do rejestru pomocniczego *RP*. Następnie *UPP* analizuje wartości δ_0 δ_1 słowa *D* w *RP* oraz wartości γ_0 γ_1 słowa *C* i zgodnie z ta-

blicą 2.2.1 pobiera albo nie pobiera z pamięci *P* odpowiednie wielkości i podstawia do słowa *C* w *RP*.

Podobnie postępuje się dla słowa *B*, a następnie słowa *A*. Potem przekazuje się zawartość *RP* do pamięci buforowej. Liczniki pamięci buforowej *LPB* sterują przesyłaniem zawartości *RP* do zwalniających się miejsc *PB* oraz pobieraniem kolejnych instrukcji z *PB* i przekazywaniem słów *C* i *D* tej instrukcji do rejestru operacyjnego *RO*, zaś słów *A* i *B* do arytmometru *AR*. Zależnie od słowa *D* w *RP* w *AR* jest wykonywane odpowiednie działanie, zaś wynik zostaje przesłany pod adres zawarty w słowie *C* w RO. Powyższy opis dotyczył instrukcji pierwotnych. Dla instrukcji wtórnej wykonuje się podstawianie według tabeli 2.2.1., następnie przesyła słowa *A* i *B* z *RP* do arytmometru adresów *AA*. W dalszym ciągu wykonuje się działanie określone przez słowo *D* w *RP* i przesyła wynik pod adres zawarty w słowie *C* w *RP*.

W przypadku instrukcji wtórnej przesyłania zespołowego następuje, po przygotowaniu słów *A,B,C* według tablicy 2.2.1, przesłanie argumentów z miejsc wyznaczonych przez słowa *A* i *B* do miejsc wyznaczonych przez *C*. Przesyłaniem steruje *UPP*, który z kolei jest sterowany zawartością *RP*.

Dla instrukcji wtórnej warunkowej następuje zbadanie znaku różnicy adresów ze słów *A* i *B* w *RP* (korzystając z pomocy *AA*) i przesłanie albo nieprzesłanie zawartości słowa *C* z *RP* do *L*.

Instrukcja pierwotna warunkowa zależnie od znaku różnicy liczb ze słów *A* i *B* tej instrukcji przesłanych z *PB* do *AR* powoduje przesłanie albo nieprzesłanie zawartości słowa *C* z *RO* do L oraz odpowiednio wyzerowanie albo niewyzerowanie *PB*.

W instrukcji programowanej występuje przygotowanie słowa *D* według tablicy 2.2.1, przesłanie zawartości *L* i *Ł* do odpowiednich słów instrukcji, której adres znajduje się w słowie *D* w *RP*, oraz przesłanie zawrtości słowa *D* z *RP* do *L*.

Układ oznaczony na tablicy 4.2.1 przez „*LPB*" składa się z trzech części:

1. licznika *P* zliczającego mod *p* ilość wprowadzonych do *PB* instrukcyj,
2. licznika *R* zliczającego mod *p* ilość wyprowadzonych z *PB* instrukcyj,
3. licznika różnicowego *LR* wskazującego ilość niewykonanych instrukcji w *PB*

gdzie *p*-pojemność pamięci buforowej.

Stan *LR* równy zeru przerywa pobieranie z *PB* do *AR*, stan *LR* równy *p* przerywa wprowadzanie z *RP* do *PB*. Licznik *P* służy do sterowania napełnianiem cyklicznie następujących po sobie miejsc w *PB*. Stan licznika *P* i *LR* jest zwiększony o 1 przy każdorazowym wprowadzaniu instrukcji do PB.

Licznik *R* służy do sterowania pobieraniem instrukcji z *PB* z miejsc następujących po sobie cyklicznie. Przy pobraniu instrukcji z *PB* stan licznika *R* jest zwiększany, zaś licznika *LR* zmniejszany o 1. Wykorzystując zawartość liczników można poszczególne słowa w *RP* i *PB*, uczynić dostępnymi poprzez ogólny układ adresowy. Dałoby to zmniejszenie ilości pobrań i przesłań z ew. do pamięci głównej[51], oraz mogłoby być wykorzystane do przesyłania parametrów wywoływanym programom[52]. Ze względu na oczywistość rozwiązania technicznego nie wydaje się celowe podawanie tu szczegółowego opisu.

Jednoczesna praca układu pobierającego liczby z pamięci[53] i arytmometru liczb sprawia, że zmniejsza się czas wykonywania instrukcyj pierwotnych w porównaniu do czasu wykonywania odpowiednich instrukcji w zwykłej maszynie.

Stworzenie warunków do jednoczesnej pracy arytmometru adresów z arytmometrem liczb daje możliwość zmniejszenia ogólnego czasu liczenia danego zagadnienia. Występujące w niektórych instrukcjach, na skutek wielokrotnego pobierania, zwiększenie czasu pobrania określonej wielkości z pamięci ma przeciwwagę w postaci zmniejszenia ilości instrukcji potrzebnych do ułożenia programu określonego problemu.

Jasne jest, że w maszynie wyposażonej w szybką pamięć główną pamięć buforowa ma mniejsze zastosowanie[54], ale i tutaj podane wyżej zasady mają wpływ na prędkość liczenia.

4.2. WYKORZYSTANIE PAMIĘCI

Wydaje się pożądane zwrócenie uwagi na czynniki decydujące o wykorzystaniu pamięci przy użyciu wyprowadzonego kodu instrukcyjnego. To samo słowo w instrukcji może zawierać zapis liczby albo adres. Z tego wynika, że nie będzie zbędnych cyfr w instrukcji (gdy słowa będą zawierały adresy), jeżeli ilość cyfr w liczbie będzie zbliżona do ilości cyfr w adresie. Wskazuje to na największą przydatność kodu dla zadań wymagających małej dokładności a jednocześnie dużej pojemności pamięci. Przegląd zbudowanych maszyn wskazuje na istnienie maszyn o długości słowa liczbowego 15^{55}, 16^{56}, 17^{57}, 20^{58} cyfr binarnych oraz adresu składającego się z 13^{59}, 15^{60}) cyfr binarnych. W przyjętym przez nas systemie adresowania

[51] Liczbę dostarczoną już do *PB* można by pobierać z *PB* do *RP*, tak samo wynik działania w AR przesyłać do *PB* — jeżeli wchodzi jako składnik do następnego działania.
[52] Zwłaszcza dla programów z małą ilością parametrów.
[53] Nie licząc tego, ze część liczb znajduje się w instrukcji.
[54] Vid. [8] str. 429.
[55] Whirlwind I [35] str. 345.
[56] USAF [42].
[57] EDSAC [50].
[58] C. S. I. R. O. Mark I [27].
[59] ERA 1101 [31].
[60] I. B. M. 705.

adres składa się z części $\sigma_2 \ldots\ldots \sigma_k$ numerującej całą pamięć i części $\sigma_{k+1} \ldots \sigma_n$, dla której wystarczy znacznie mniejsza ilość cyfr, np. 7. Widać z tego, że adres miałby dla tych maszyn długość zbliżoną do długości liczby, np. dla pamięci o pojemności równej pojemności pamięci ERA 1101 (adres = 13 i odstęp adresowy = 7 cyfrom binarnym) adres miałby długość 20 cyfr binarnych (nie licząc cyfr $\sigma_o\, \sigma_1$).

Jak łatwo zauważyć, chociażby z podanych przykładowo programów, różnica w pojemności pamięci potrzebnej na programy w proponowanej i zwykłej metodzie zależy także od ilości stałych i występujących tylko w jednym miejscu programu wartości zmiennych (odpowiada występowaniu tylko jeden raz symbolu danej zmiennej w formule matematycznej).

5. DODATEK

5.1. KOD INSTRUKCYJNY O ZMIENNEJ DŁUGOŚCI INSTRUKCJI

Celem wyeliminowania straty na pojemność pamięci dla maszyn pracujących ze słowem liczbowym o długości różniącej się od długości adresu rozpatrzymy koncepcję kodu o zmiennej długości instrukcji. Aby uniknąć zbędnego pobierania i przesyłania występującego w kodzie trójadresowym, instrukcja będzie miała ponadto zmienną ilość adresów[61].

Niech każda instrukcja składa się z pewnej, teoretycznie dowolnej, ilości krótkich słów, np. słów o długości potrzebnej dla oznaczenia wszystkich miejsc w pamięci.

Jedno (względnie więcej) słowo określa:

1) rodzaj instrukcji (pierwotna, programowana, ...).

2) ilość argumentów, które należy pobrać z pamięci oraz przesłać do pamięci[62] (resztę składników do bieżącej instrukcji zawiera wówczas arytmometr).

3) operację (np. dodawanie).

Słowa zawierające powyższe informacje tworzą część operacyjną instrukcji. Następne słowa instrukcji są zajęte przez liczby wchodzące do bieżącej operacji (liczba jest utworzona z kilku słów) względnie przez ich adresy (wystarczy wtedy odpowiednio mniejsza ilość słów), oraz przez adres wyniku (jeżeli wynik działania nie pozostaje w arytmometrze). Każde słowo instrukcji posiada swój adres[63]. W tablicy 5.1. pokazano przykładowo instrukcję $\overline{m + 5}$ z programu w tablicy 3.1.1. przedstawioną w kodzie o zmiennej długości.

[61] Vid. także [34].

[62] Odpowiada ilości adresów w instrukcji podanej w [34] str. 2. Ilość składników będących przedmiotem działania bieżącej instrukcji jest teoretycznie biorąc dowolna.

[63] Adresem liczby utworzonej z kilku słów jest adres pierwszego słowa.

Tablica 5.1

Przykład instrukcji w kodzie o zmiennej długości

A_1			B_1	C_1	D_1
$n+5$	$n+4$	$n+3$	$n+2$	$n+1$	n
0			Y'_{n+1}	Y'_n	O_j

Pierwsze słowo (o adresie n) wskazuje rodzaj instrukcji (w danym przypadku pierwotna), ilość "adresów" (trzy) i operację (dodawanie). Słowo drugie (o adresie $n + 1$) zawiera adres wyniku (oznaczony przez Y_n'). Zawartość słów $n + 3$, $n + 4$, $n + 5$ tworzy liczbę 0.

Części instrukcji oznaczone przez D_1; C_1; B_1; A_1; odpowiadają, oczywiście słowom D, C, B, A.

Część operacyjna instrukcji D_1, jak już wspomniano, składa się z jednego lub więcej słów. Zawartość pierwszego słowa sygnalizuje, między innymi, z ilu słów składa się D_1.

Część C_1[64] (o długości jednego słowa przy zastosowaniu zwykłego systemu adresowania, zaś o długości dwóch słów przy systemie opisanym w 2.2[65] zawiera adres wyniku.

Części A_1 i B_1 zawierają liczby wchodzące do operacji względnie ich adresy[66]. Jasnym jest, że np. instrukcja programowana może dotyczyć większej ilości argumentów i wówczas będzie ona odpowiednio dłuższa. Opisany schematycznie kod łączy zalety kodu jedno, dwu i trójadresowego a jednocześnie wprowadza ekonomiczne wykorzystanie pojemności pamięci.

Korzystanie z pamięci w zwykłej maszynie można podzielić na:

a) pobranie instrukcji,
b) pobranie słowa z miejsca o stałej zawartości,
c) pobranie słowa z miejsca o zmiennej zawartości,
d) przesłanie słowa do miejsca pamięci o zmiennej zawartości.

Analogiczne korzystanie z pamięci występuje i w naszym kodzie, z tym, że pobieranie opisane pod (a) i (b) tworzy jednolitą całość (czas potrzebny na to pobranie może być znacznie zmniejszony jedną z metod eliminacji czasu oczekiwania na instrukcję opisanych w [8]). Średni czas oczekiwania dla punktu c można znacznie zmniejszyć przy pomocy pamięci buforowej instrukcji[67]). Należy jednak zwrócić uwagę, aby w pamięci bu-

[64] Gdy wynik pozostaje w arytmometrze, to C_1 nie istnieje.

[65] System adresowania może być sygnalizowany wartością odpowiednich cyfr w pierwszym słowie części C_1.

[66] Stosuje się również podana na stronie 55 uwaga o sygnalizacji systemu adresowania.

[67] Vid. także str. 103 o przynoszeniu wartości niektórych zmiennych łącznie z instrukcją.

forowej nie znajdowały się wielkości, które w międzyczasie ulegają zmianie.

Przedstawione w [16] wyniki obliczeń (dla kodu jednoadresowego), dotyczące częstotliwości występowania poszczególnych instrukcji, podają, że przesyłanie typu wymienionego w pkt. d (instrukcja *EC* w kodzie maszyny MANIAC) występuje tylko w ok. 12% instrukcji.

Powyższa analiza wskazuje, że wprowadzony kod umożliwia znaczną redukcję udziału czasu korzystania z pamięci w ogólnym czasie liczenia, ponieważ i przesyłań do pamięci jest stosunkowo niewiele.

Aby uprościć adresowanie, które ze względu na zmienną długość instrukcji będzie raczej utrudnione, można zastosować zmienny system adresów[68]. W związku z tym warto także zauważyć, że zgodnie z przeprowadzonymi przez Pearceya[69] badaniami wykorzystania pamięci, jedynie około 5% pamięci powinno stanowić pamięć wymazywalną[70], tzn. tylko zawartość słów tworzących powyższe 5% ulega zmianie w trakcie liczenia, czyli tylko niewielka część słów podlega adresowaniu.

5.2. TERMINOLOGIA

Adres — numer przyporządkowany rejestrowi lub miejscu w pamięci.

Adres względny — zwykle adres instrukcji lub argumentu przy założeniu, że pierwsza instrukcja programu lub pierwszy argument zespołu argumentów posiada adres równy zeru. U nas adres instrukcji przy założeniu, że bieżąca instrukcja ma adres równy zeru.

Arytmometr — zespół w maszynie cyfrowej wykonujący operacje arytmetyczne i logiczne.

Biblioteka programów — kolekcja dla danej maszyny standartowych i sprawdzonych programów, za pomocą której można rozwiązać wiele typowych zadań i części zadań.

Instrukcja — zespół znaków, który wyznacza operację łącznie z jednym lub więcej adresem i który jako całość powoduje odpowiednie działanie maszyny na wskazanych argumentach.

Instrukcja n-adresowa — instrukcja zawierająca najwyżej n-adresów argumentów (np. $n = 1, 2, 3$).

Kod instrukcyjny — lista wszystkich instrukcji wykonalnych przez daną maszynę cyfrową.

Licznik instrukcyj — część maszyny cyfrowej zapewniająca kolejne wykonywanie instrukcji danej sekwencji. Zmiana zawartości licznika instrukcyj odpowiada przejściu do nowej sekwencji.

[68] "floating adress system".
[69] Vid. [30].
[70] "erasable".

Pamięć — urządzenie, do którego można wprowadzać informacje oraz wyprowadzać je później. Z pamięci można pobierać wielokrotnie tę samą informację. Przesłanie nowej informacji do określonego miejsca pamięci niszczy jednocześnie starą zawartość tego miejsca.

Pamięć buforowa — pamięć przechowująca kilka najbliższych do wykonania instrukcji i napełniana w miarę ich wykonywania następnymi instrukcjami.

Parametr programu — parametr wprowadzany do programu podczas liczenia (przez inny program).

Pętla — powtórzenie sekwencji instrukcji w programie.

Pobieranie — proces wydawania przez pamięć lub rejestr duplikatu argumentu znajdującego się w danym miejscu pamięci lub rejestru. Pobieraniu oczywiście stale towarzyszy umieszczenie (z modyfikowanego niekiedy) duplikatu w wyznaczonym bieżącą instrukcją miejscu maszyny[71].

Program interpretacyjny — program analizujący program wyrażany w danym kodzie i inicjujący wykonanie wskazanych operacji za pomocą odpowiednich programów (blok programów funkcji).

Programowanie optymalne — programowanie w taki sposób, że jest potrzebny minimalny czas oczekiwania dla otrzymania informacji z pamięci.

Rejestr — urządzenie dla przechowywania jednego lub więcej słów.

Słowo — ciąg $n + 1$ cyfr.

5.3. SYMBOLIKA

$\alpha_0 \dots \alpha_n$ — kolejne cyfry słowa A
$\beta_0 \dots \beta_n$ — kolejne cyfry słowa B
$\gamma_0 \dots \gamma_n$ — kolejne cyfry słowa C
$\delta_0 \dots \delta_n$ — kolejne cyfry słowa D

A, B, C, D — poszczególne słowa instrukcji

$\Rightarrow$ — symbol przesłania w instrukcji pierwotnej
$\rightarrow$ — symbol przesłania w instrukcji wtórnej

[71] Wariant pobierania występujący w pracy omówiono w 2.2.

Fh — symbol h-ej instrukcji programowanej

$\tau_0 \ldots \tau_n$ — kolejne cyfry słowa X (gdzie $X = A, B, C$)

$\sigma_0 \ldots \sigma_n$ — kolejne cyfry dowolnego słowa instrukcji

$t_0 \ldots t_n$ — kolejne cyfry słowa o adresie utworzonym przez cyfry $\sigma_2 \ldots \sigma_k$

$i_0 \ldots i_n$ — kolejne cyfry dowolnego słowa

P — pamięć

RP — rejestr pomocniczy

AR — arytmometr

UPP — układ pobierająco — przesyłający

L — licznik instrukcyj

a' — adres I rzędu

a'' — adres II rzędu

a''' — adres III rzędu

$\overline{a}$ — adres instrukcji

$\left.\begin{matrix}\overline{a}\,A\\ \overline{a}\,B\\ \overline{a}\,C\\ \overline{a}\,D\end{matrix}\right\}$ — adresy słów A, B, C, D. instrukcji

a — adres względny instrukcji

$\left.\begin{matrix}\overline{a}\,A\\ \overline{a}\,B\\ \overline{a}\,C\\ \overline{a}\,D\end{matrix}\right\}$ — adresy słów A, B, C, D instrukcji o adresie $\underline{a}$

$\overline{(\ )}$ — słowo A, B, C ew. D instrukcji, które będzie ulegać zmianie w trakcie wykonywania programu

$[a]$ — zawartość słowa o adresie a

$Ao_i B \Rightarrow C$ — oznaczenie instrukcji pierwotnej

$Ao_j^* B \rightarrow C$ — oznaczenie instrukcji wtórnej

Z — adres słowa zawierającego własny adres III-go rzędu

S — adres akumulatora

PB — pamięć buforowa

AA — arytmometr adresów

RO — rejestr operacyjny

LPB — licznik pamięci buforowej

Zakład Aparatów Matematycznych
Instytutu Matematycznego PAN.

WYKAZ LITERATURY

[1] Alway, G. G.: *Optimum coding, Symposium of Automatic Digital Conference, N. P. L.* 1953, 65—70.

[2] Böhm, C.; *Calculatrices digitales. Du déchiffrage de formules logico — mathématiques par la machine même dans la conception du programme, Annali di Matematica, cz.* II 1954, 175—217.

[3] Brocker, R. A., Wheeler D. P.: *Floating aperations on the EDSAC, Math. Tables and other Aids to Computation,* 1953, 7 *Nr* 41, 37—47.

[4] Coombs, A. W. M., Mosaic.: *Symposium of Automatic Digital Conference, N. P. L.* 1953, 38—44.

[5] Coombs, A. W. M., Mosaic.: *An Electronic Digital Computer, Part* 2, *The Post Office Electrical Engineers Journal, Vol.* 18, *April* 1955, 137—141.

[6] Curry, H. B.: *The logic of program composition, Applications scientifiques de la logique mathématique, Actes du* 2-*e Colloque international de logique mathématique Paris,* 25—30 *Aout* 1952.

[7] Clippinger, R. F., Dimsdale B., Levin J. H.: *Automatic digital computers in industrial research, J. Soc. Indust. Appl. Math.* 1953, 1954.

[8] Freedman, A. L.: *Elimination of waiting time, Proc. Comb. Ph. Soc. Vol.* 50, *part* 3, *July* 1954, 426—438.

[9] Gill, S.: *The diagnosis of mistakes in programmes on the EDSAC, Proc. Roy. Soc., London A* 206 1951, 538—554.

[10] Gill, S.: *Getting programmes right, Symposium of Automatic Digital Conference, N. P. L.* 1953, 80—83.

[11] Goldstine, H. H., Neumann von J.: *Preliminary discussion of the logical design of an electronic computing instrument.*

[12] Goldstine, H. H., Neumann vo'n J.: *Planning and coding of problems for an electronic computing instrument, cz.* II,

[13] Greig, J.: *The Circle computer, MTAC,* 1953, 7 *Nr.* 44 249—255.

[14] Grenwald, S., Haueter R. C.: Alexander S. N., SEAC, *Proc. Inst. Radio Engrs,* 1953, 41, *Nr.* 10, 1300—1313.

[15] Hartree, D. R.: *Numerical Analysis, Oxford at the Clarendon Press* 1952, 255—273.

[16] Herbst, H. Metropolis N., Wells M. B.: *Analysis of problem codes on the MANIAC, MTAC* 9, 1955. 14—20.

[17] Hopper, G. M., Automatic coding for Digital Computers, Computers and Automation, Vol. 4, Nr 3, 1955, 21—24.

[18] Hopper, G. M., Mauchly J. W.: *Influence of programming techniques on the design of computers, PIRE,* 1953, 1250—1254.

[19] Hume, J. N. P.: *Input and organization of subroutines for Ferut, MTAC* 8, *Nr.* 45 1954, 30—36.

[20] Huskey, H. D.: *Characteristics of the Institute for Numerical Analysis Computer, MTAC* 4, 1950, 103—108.

[21] Koons, L., Lubkin S.: *Conversion of Numbers From Decimal to Binary Form in the EDVAC* 3, 1949, 426—431.

[22] Kilburn, T., Tootill C. C., Edwards D. B. G., Pollard B. W.: *Digital Computers at Manchester University, Proc. J. E. E. Vol.* 100, *part* II, 1953, 487—501.

[23] *Kratkoje Opisanie BESM, maszynopis.*

[24] Mac-donald, N.: A Big Inventary Problem and the IBM 705, Computers *and Automation,* 1955, 9, 6.

[25] Mutch, F. N.: *Conversion routines, Symposium of Automatic Digital Conference, N.P.L.* 1953, 74—80.

[26] Pawlak, Z.: *Bezrozkazowa maszyna cyfrowa, maszynopis.*

[27] Pearcey, T., Hill, G. W.: *Programme design for the C.S.I.R.O. Mark* I *computer.* I *computer conventions. Austr. J. Phys.* 1953, 6, *Nr* 3, 316—334.

[28] Pearcey, T., Hill, G. W., *Programme design for the C.S.I.R.O. Mark* I *Computer* II, *Programme techniques. Aust. J. Phys.*, 1953, 6, *Nr.* 3, 335—356.

[29] Pearcey, T., Hill, G. W.: *Programme design for the C.S.I.R.O. Mark* I *Computer., III, Adaptation of routines for elaborate arithmetical operations, Aust. J. Phys.* 1954, 485—504.

[30] Pearcey, T., Hill, G. W., Ryan R. D.: *The effect of interpretive techniques on functional design of computers, Aust. J. Phys.* 1954, 506—519.

[31] Perry, D. P.: *Minimum Access programming, MTAC* 6, 59 1952, 172—182.

[32] *Programming Research Section, Automatic Programming the A* 2 *Compiler System Part 1, 2, Computers and Automation Vol.* 4, *Nr* 9, 10 1955.

[33] Poel, van der W. L.: *Dead programmes for a magnetic drum automatic computer, Appl. Scient. Res.*, 1953, *B* 3, *Nr* 3 190—198.

[34] *Proposed code for ICCE* II, *Imperial College of Science and Technology, Departament of Mathematics*, 1—35.

[35] Richards, R. K., *Arithmetic operations in digital computers, D. Van Nostrand Company, Inc. N. York* 1956.

[36] Rutishauser, H.: *Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen*, 1952, 1—45.

[37] Rutishauser H., Speiser A., Stieffel E.: *Programmgesteuerte digitale Rechengeräte*, I, II, II, IV, *Z. Angew Math. Phys* 1, 1950, 277—297, 339—362, 2, 1951, 1—25, 63—92.

[38] Samelson, K., Bauer F. L.: *Massnahmen zur Erzielung kurzer und übersichtlicher Programme für Rechenautomaten, Z.A.M.M. Bd.* 34, *Nr* 7 1954, 262—272.

[39] Schecher, H.: *Vorschläge zur Verbesserung des Adressenrechenwerks bei der PERM, Arbeitsgruppe für Elektronische Rechenanlagen Techn. Hochschule München, Mitteilung N.* 251, *Reihe M. Januar* 1956.

[40] Shyder, F. E., Livingston A. M.: *Coding of a Laplace Boundary Value Problem for the UNIVAC, M.T.A.C.* 3, 1949. 341—350.

[41] Staff, M. D. L.: *The incorporation of subroutines into a complete problem on the NBS eastern automatic computer., MTAC* 4, 1950, 164—168.

[42] Stone, J. J.: *The USAF-Fairchild specialized digital computer., MTAC*, 1953, 7, *Nr* 41, 35—37.

[43] Swire, B. E.: *A proposed modification to the C.S.I.R. O. Mark* I *Computer, Austr. J. Phys.* 1955, *Vol.* 8 *No* 1, 184—186.

[44] Stringer, J.: *Microprogramming and the choice of order code, Symposium of Automatic Digital Conference N.P.L.* 1953, 71—74.

[45] *Stroje na zpracování informaci Sbornik* I, *ČAV, Praha* 1953 *e.*

[46] Thomas, H.: *Fundamentals of Digital Computer Programming, PIRE* 1953/III, 1245—1249.

[47] Tocher, K. D.: *Report on the work of the computer group. The Imperial College of Science and Technology*, 1952, 1—125.

[48] Walker, J. P.: *A special purpose digital computer, MTAC* 1953, 1, *Nr* 43, 140—195.

[49] Wheeler, D. J.: *Programme organization and initial orders for the EDSAC, Proc. Rov. Soc. London Ser A.* 202, 573—589, 1950.

[50] Wilkes, M., Wheeler D. J., Gill, S.: *The preparation of programs for an electronic digital computer, Addison-Werley Press* 1951, 1—169.

[51] Wilkes, M. V.: *The use of a „floating adress" system for orders in an automatic digital computer, Proc. Cambridge Phil. Soc.* 49, 84—89 1953.

[52] Wilkes, M. V.: *Programme design for a high speed automatic calculating machine, J. Scientific Instrument* 26, 1947, 217—220.

[53] Wilkes, H. V., Stringer J. B.: *Micro-programming and the design of the control circuits in an electronic digital computer, Proc. Cambridge Phil. Soc.* 1953, 49, 230—238.

[54] Wilkinson, J. H.: *An Assesment of the system of optimum coding used on the Pilot ACE at the N.P.L. Phil. Trans. Roy. Soc. London Ser. A, Nr* 946, *Vol.* 248. 253—281, 1955.

В. ЯВОРСКИ

ПРОГРАММИРОВАННАЯ ВЫЧИСЛИТЕЛЬНАЯ МАШИНА ДИСКРЕТНОГО ДЕЙСТВИЯ С НОВЫМ ПОСТРОЕНИЕМ ИНСТРУКЦИИ

Резюме

Вычислительные машины дискретного действия проектированные в начале их развития имели отдельные запоминающие устройства инструкции и аргументов. Следующим этапом развития вычислительных машин было строительство машин с общим запоминающим устройством для инструкции и аргументов. В этом устройстве однако инструкции и аргументы выступают раздельно. Следуя по этому пути в настоящей работе сделана попытка разработать новую идею вычислительной машины дискретного действия с иннструкционным кодом, в котором аргумент может составлять часть инструкции[1]. Одновременно дана схема системы адресовки и чтения замещающая применяемые до сих пор методы адресовки и модифицирования инструкции, осуществляемые при помощи специальных устройств и программ. Новое построение инструкции влечет за собой так изменение структуры вычислительной машины, как и существенные изменения в программировании. Эти изменения выступают не только при составлении программ из одиночных инструкций и в адресной системе, но тоже в пользовании готовыми библиотечными программами (при помощи введенной программированной инструкции).

Программированная инструкция вместе с обязательной для неё схемой адресовки и чтения обеспечивает значительное упрощение составления программ из библиотечной программы и дает подобные облегчения, как интепретационная техника.

---

[1] Идею введения нового построения инструкции дал анализ методов применяемых для ускорения исчисления в вычислительных машинах с задерживающим запоминающим устройством, а особенно анализ методов оптимального программирования.

Предлагаемый инструкционный код уменьшает количество операции чтения и пересылания к запоминающему устройству (вследствие существования аргументов в инструкциях и инструкции группового пересылания), что особенно важно для машин с большим временем выбора.

Введенный в настоящей работе перечень инструкции не является оптимальным. По потребности этот перечень может быть модифиюирован и пополнен.

Хотя в работе не поданы подробные схемы и технические решеня, то однако автор старался обеспечить возможность простой реализации и уменьшения времени необходимого для пользования запоминающим устройством в общем времени исчисления.

Анализ использования запоминающего устройства дал побуждения для разработки инструкционного кода переменной длины инструкции. Кажется целесообразным подчеркнуть большие возможности заключены в этом коде.

Основные тезисы настоящей работы были доложены в 1955 году на семинаре Лаборатории математических аппаратов Математического института Польской Академии Наук.

W. JAWORSKI

A PROGRAMMED DIGITAL COMPUTER WITH A NEW INSTRUCTION STRUCTURE

Summary

The early digital computers were supplied with separate stores of instructions and operands. The design of computers with a commen store for instruction and operands was the next step of developement. The instructions and operands in this store were, anyway, separated. The paper tries to go further this way, working out an idea of the digital computer with instruction code, in which the operand can be a part of the instruction[1]. A scheme of addressing and taking — out is also presented, replacing the usually applied methods of adressing and modificating the instruction, based on special arrangements, instructions or programmes. The new instruction structure was followed by a change in the structure of the digital computer, as well as in its programming. The changes occur not only in forming the provides a significant simplification of the programme forming from the library subroutines (it is performed with a so called programmne instruction). of library subroutines (it is performed with a so called programmne instruction). The programme instruction together with its addressing and taking — out system provides a significant simplification of the programme forming from the library subroutines and gives facilitations similar to those obtained with the interpretive technique. The proposed instruction code reduces the quantity of taking-out and transferring to the store operations (owing to the operands contained in instructions and instruction of combined transferring. This property is of great importance to computers with a large access time store. The list instructions introduced in this paper is not optimum. If necessary, this list can be modified and supplemented. Although no detailed diagrams and technical solutions are presented, efforts were made to acertain the possibility of simple technical realization and of reducing the time necessary to make use of the store, as referred to overall computing time. An analysis of the store effective work was a motive of working out an instruction code with variable instruction lenght, described here in general outlines. It seems advisable to emphasize the great possibilities contained in this code. The basic ideas of this paper were reported on a seminary in the Laboratory of Mathematical Machines of the Polish Academy of Science, in 1955.

[1] The idea of introducing a new instruction structure appeared by the analysis of the methods of shortening computing time in computers with delay storage, and especially by the analysis of optimum coding methods.