ZASTOSOWANIA MATEMATYKI
V (1961)

С. ПАШКОВСКИЙ (Варшава)

ВНЕШНИЙ КОД ДЛЯ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

1. Введение. Любой цифровой вычислительной машине свойственны команды, которые она может исполнять. Систему этих команд называют внутренним кодом машины. Обычно внутренний код состоит только из самых простых и необходимых команд — логических, арифметических, управляющих и прочих. Более сложные операции, как например вычисление значений элементарных функций, надо в программе расчленять на эквивалентные им группы команд внутреннего кода. Программирование в таком коде длинно и неудобно, поэтому вне машины часто пользуются внешним кодом, т. е. другой системой операции, любая из которых тождественна определенной группе команд внутреннего кода. Эти обобщенные команды иногда являются очень сложными (нахождение всех нулей алгебраического полинома, перемножение двух матриц и т. д.). Они по своей форме ближе обычному описанию задачи, чем команды внутреннего кода, тем более, что к специальным задачам (например, к автоматическому переводу) можно применять особые коды. Кроме того, внешний код, общий для нескольких цифровых машин, делает возможным обмен программ между вычислительными центрами.

Операции внешнего кода „не понятны" машине, перед исполнением их надо перевести на язык внутреннего кода. Такой перевод может выполнить сама машина с помощью программирующей или интерпретирующей программы. Мы будем эти программы, действующие на программу, построенную во внешнем коде, обозначать соответственно символами ПП и ИП. Существенное различие между ПП и ИП заключается в том, что ПП переводит всю программу на внутренний код, после чего счет ведется уже обычным образом, согласно переведенной программе. В случае надобности полученную программу во внутреннем коде можно тотчас же вывести из памяти машины и использовать в другое время. ИП ставит в соответствие отдельным

операциям внешнего кода, не изменяя их, эквивалентные группы команд внутреннего кода с целью их немедленного исполнения, так что нельзя отделить обработку программы от ее выполнения.

Род программы, с помощью которой мы намереваемся обрабатывать внешний код, налагает на него определенные условия, так что ПП и ИП применяются к разным внешним кодам. Несмотря на это мы можем эти программы сравнивать с точки зрения времени и объема памяти, необходимых для использования внешнего кода. Обработка программы с помощью ПП обычно довольно сложна, но делается только один раз. Во время счета в памяти должна находиться переведенная программа во внутреннем коде, которая часто длиннее подобной программы, составленной непосредственно в этом коде опытным программистом. С другой стороны, в случае использования ИП в памяти все время должна быть сама ИП и обрабатываемая программа во внешнем коде. Обработка операций с помощью ИП, происходящая каждый раз, когда операцию надо исполнить, обычно проста. В общем, выбор внешнего кода и связанной с ним обрабатывающей программы (ПП или ИП) определяется в значительной степени основными параметрами цифровой машины, характером программируемых задач и наконец тем, насколько часто повторяются однотипные задачи.

Настоящая работа содержит описание довольно примитивного внешнего кода, предназначенного для программирования задач численного анализа. Программу, построенную в этом коде, сначала подвергают обработке с помощью простой ПП, в результате чего получается эквивалентная программа в некотором промежуточном коде, связанном уже с особенностями конкретной вычислительной машины. После этого счет ведется с помощью ИП и набора стандартных подпрограмм, определяющихся характером решаемой задачи. Описываемый нами внешний код предназначен главным образом для машин, обладающих сравнительно небольшой скоростью и оперативной памятью. Это заставило нас отказаться, например, от введения в состав кода обычных арифметических формул. Во внешнем коде есть много общего с типичными внутренними кодами известных машин. Он отличается от таких кодов прежде всего: 1° — возможностью расширения системы операций кода без его изменения, 2° — переменным количеством адресов, 3° — более гибким способом адресации слов и чисел, 4° — особой структурой циклов в программах. К сожалению автор не мог всесторонне испытать своего проекта на практике и этот проект является только эскизом кода.

2. Внешний код. Программа во внутреннем коде расчленяется на команды, которые в свою очередь состоят обычно из кода команды

и одного, двух или трех адресов. Понятиям команды и ее составных частей соответствуют во внешнем коде понятия блока и его сегментов (хотя, как мы увидим позже, блоки не исчерпывают еще всей программы). Более или менее общая операция, какой является блок, зависит от некоторого числа аргументов и определяет значения некоторых величин. Род операции выполняемой блоком, аргументы или адреса аргументов и адреса результатов операции определяются отдельными сегментами блока. Порядок и число сегментов в блоке любого типа всегда фиксированы.

Мы различаем сегменты шести типов, обозначенных соответственно буквами α, β, γ, δ, ε и ζ.

Тип α. Код блока — трехбуквенный символ, взаимно однозначно сопоставлен типу операции, исполняемой блоком, например SUB (вычитание), SQR (извлечение квадратного корня), UNJ (безусловная передача управления) и т. д.(1). Первый сегмент любого блока является всегда сегментом типа α и всегда обозначает код этого блока. Если какой-нибудь другой сегмент блока B тоже является кодом, то это код блока, служащего в определенном смысле аргументом для блока B.

Мы предполагаем, что решение программируемой задачи непосредственно требует чисел только одного рода, например чисел с плавающей запятой, так что в принципе только такие числа являются начальными данными задачи и ее промежуточными и окончательными результатами. Адреса таких чисел являются сегментами типа β. Эти адреса и некоторые другие величины представляют собой второй класс чисел — целые числа, под которые отведена отдельная часть памяти.

Тип β. Внешний адрес числа (с плавающей запятой) — целое неотрицательное трехзначное число, одно из чисел $000, 001, \ldots, 999$ (знак плюс опускается).

Для обозначения содержимого какой-нибудь ячейки памяти мы будем заключать ее символ (адрес или номер) в круглые скобки. Помещение символа N в ячейку r изображается формулой $N \to r$. Пользуясь этими обозначениями мы приведем сейчас примеры простейших арифметических блоков, перечисляя для каждого его общий вид, название и выполняемую им операцию:

1° ZER r, стирание, $0 \to r$,

2° TRA pr, перенос, $(p) \to r$,

3° SQUpr, возведение в квадрат, $(p)^2 \to r$,

---

(1) Все коды блоков в этой работе являются сокращениями английских названий соответствующих операций.

4° ADDnpr, суммирование, $(n)+(p) \to r$,
5° SUBnpr, вычитание, $(n)-(p) \to r$,
6° MULnpr, умножение, $(n)\cdot(p) \to r$,
7° DIVnpr, деление, $(n):(p) \to r$.

Буквы n, p и r обозначают сегменты типа β, т. е. адреса чисел. В блоках ZER и TRA это могут быть также сегменты типов γ и ε, т. е. номера сегментов и адреса регистров модификации.

Тип γ. Номер сегмента — символ, состоящий из буквы S и целого неотрицательного двузначного числа (без знака плюс), например S12. Нумеруются только те сегменты, содержимое которых упоминается в программе, как аргумент или результат какого-нибудь блока. Порядок этих номеров никак не связан с порядком блоков в программе, так что по отношению к блокам реферируемый внешний код является безадресным. Иногда нужен номер блока (например того, которому другой блок передает управление). Номер блока по определению равен номеру первого сегмента этого блока.

Как во внутренних кодах большинства цифровых машин, так и в нашем внешнем коде блоки выполняются в таком порядке, в каком они написаны в программе. Этот естественный порядок можно в случае надобности изменить с помощью блока условной или безусловной передачи управления (условного или безусловного перехода). Вот примеры применения таких блоков: UNJ S12 — безусловная передача управления блоку, начинающемуся с сегмента S12;

(1) EQJ 000 001 S00

— сравнение чисел, расположенных в ячейках 000 и 001, и передача управления 1° блоку, начинающемуся с сегмента S00, если (000) = = (001), 2° блоку, следующему за блоком (1), если (000) $\neq$ (001). Возможны, конечно, и более сложные блоки передачи управления.

Применение арифметических блоков и блоков передачи управления поясняется программой I, помещенной вместе с остальными примерами в конце работы. Программа вычисляет значение функции

$$f(x, y) = \begin{cases} e^x & \text{для} \quad x = y, \\ \dfrac{e^x \sin(x-y)}{x-y} & \text{для} \quad x \neq y \end{cases} \tag{2}$$

в предположении, что (000) = x, (001) = y и что это значение должно получиться в ячейке 002. Ячейка 003 используется в качестве рабочей. Кроме уже приведенных блоков в программе употребляются:

1° — блоки вычисления значений показательной функции (код EXP) и синуса (код SIN), построенные по образцу блока SQU, 2° — состоящий из единственного первого сегмента STO блок остановки работы машины.

Бланк программы состоит из семи граф:

Номер сегмента	Начало	Сегмент	Конец	Модификация сегмента		
1	2	3	4	5	6	7

(в примерах неиспользованные графы мы будем опускать, не изменяя номеров оставшихся граф). Сегменты блоков вписываются в их естественном порядке в 3-ю графу, по одному сегменту в строке. Очередные блоки можно для удобства разделять горизонтальной чертой. Если номер сегмента, находящегося в 3-й графе, упоминается в каком-нибудь блоке, то этот номер мы поместим в 1-ю графу, рядом с сегментом. В программе I второй блок может передать управление блоку остановки и единственный сегмент (STO) этого блока отмечен в 1-й графе номером S00. Назначение остальных граф мы узнаем позже.

Предположим теперь, что в некоторой задаче значения функции (2) должны вычисляться многократно, причем меняются адреса аргументов и значений функции. Чтобы не повторять отрезков программы, несущественно отличающихся между собой, мы применяем особый тип сегмента и некоторые другие обозначения. Очередные сегменты блока будут обозначаться буквами A, B, C, ...

Тип δ. Символ переменной, состоящий из буквы V и буквы, которой обозначен сегмент, где находится значение переменной, напр. VC или VE.

Смысл таких сегментов мы разъясним примером. Во всех тех местах программы, где нужно вычислить значение функции f, мы помещаем выполняющий это задание обобщенный блок (названный так в отличие от элементарных блоков, с которыми мы имели дело до сих пор) EXS $\langle x\rangle\langle y\rangle\langle f\rangle$, где $\langle x\rangle$, $\langle y\rangle$, $\langle f\rangle$ — адреса типа β, соответственно чисел x, y и значения $f(x, y)$. Код обобщенного блока избирается вообще произвольно, среди символов типа α, не являющихся кодами элементарных блоков. Порядок переменных адресов в блоке тоже произволен, но фиксирован. Различие между элементарным и обобщенным блоком состоит в том, что у последнего есть своя программа во внешнем коде. Она помещается за основной программой, т. е. той, в которой употребляется обобщенный блок. В примере, связанном с функцией f, программа обобщенного блока EXS $\langle x\rangle\langle y\rangle\langle f\rangle$

получится из программы I, ссли: 1° — заменить адреса 000, 001, 002 величин x, y и f, которые в блоке EXS составляют соответственно сегменты B, C и D, символами VB, VC и VD, 2° — перед программой I поставить строку, составленную из скобки „(" во 2-й графе и кода EXS в 3-й графе, 3° — к последней строке программы I добавить скобку „)" в 4-й графе. В итоге получится программа II.

Мы уже познакомились с двумя типами ячеек памяти, предназначенных соответственно для хранения программы и чисел (начальных данных, промежуточных и окончательных результатов). Еще иные и по-другому адресованные ячейки играют роль регистров модификации адресов и целых чисел. Эти регистры употребляются прежде всего в циклических программах, что подробно опишем ниже. Любой регистр может содержать целое трехзначное число, снабженное знаком, например −001 или +236.

Тип ε. Внешний адрес регистра модификации — символ, состоящий из буквы M и целого неотрицательного двузначного числа (без знака плюс), напр. M25.

Тип ζ. Целое трехзначное число со знаком + или − (знака + опускать нельзя, чтобы не спутать такого сегмента с адресом числа), например −196 или +000.

Для формирования сегмента или содержимого регистра модификации служит элементарный блок STRnp, выполняющий операцию $N \to p$, где N — сегмент любого типа, а p — номер сегмента или адрес регистра, подлежащего формированию. В частности, STR MUL S14 обозначает подстановку кода MUL на место сегмента S14, а STR −004 M00 — помещение числа −004 в нулевой регистр модификации.

Для изменения сегмента типа β или ζ, либо содержимого регистра модификации, т. е. для изменения некоторого целого трехзначного числа, служит блок INANpr, где N — целое трехзначное число, а p и r — номера сегментов или адреса регистров. Блок выполняет операцию $N+(p) \to r$. Например, блок INA +002 M14 M14 обозначает увеличение на два содержимого 14-го регистра.

5-я, 6-я и 7-я графы бланка программы могут содержать адреса регистров модификации, т. е. символы типа ε. На этом месте они уже не являются сегментами, а играют другую роль. Если рядом с сегментом блока B находятся адреса одного, двух или трех регистров, то это значит, что блок B выполняется не в своем истинном виде, а с увеличением упомянутого сегмента на актуальное содержимое всех этих регистров. Таким путем можно модифицировать, главным образом в циклических программах, сегменты типа β и ζ, а также заменяющий эти сегменты сегмент типа δ.

Цикл мы программируем следующим образом: 1° — прежде чем

начать цикл, мы засылаем (с помощью блока ZER или STR) нужные числа в избранные регистры модификации, 2° — сегменты, которые должны изменяться, модифицируем с помощью этих регистров, помещая их адреса в 5-ю, 6-ю и 7-ю графы бланка, 3° — после каждого повторения блоков цикла меняем в нужном направлении содержимое регистров с помощью блока INA. Сигналом начала цикла является отдельная строка, составленная из скобки „(" во 2-й графе и символа типа ζ, равного кратности цикла (числу его повторений), в 3-й графе. Эту кратность мы считаем полноправным сегментом, так что ее можно модифицировать, как любой другой сегмент типа ζ. Сигналом конца цикла является скобка „)", помещенная в 4-ю графу бланка рядом с последним сегментом последнего блока цикла.

Поясним вышесказанное примером (см. программу III). Предположим, что надо вычислить значение p полинома $a_0x^{16}+a_1x^{15}+\ldots+a_{16}$ в некоторой точке $x=\bar{x}$. При этом коэффициенты расположены в ячейках 000 (a_0), 001 (a_1), ..., 016 (a_{16}), аргумент $\bar{x}$ — в ячейке 017, а значение p полинома надо поместить в ячейку 018.

Счет ведется по рекуррентной схеме Горнера:

$$p=\big(\ldots\big((0\cdot\bar{x}+a_0)\bar{x}+a_1\big)\bar{x}+\ldots+a_{15}\big)\bar{x}+a_{16}$$

первый, лишний шаг схемы введен для сокращения объема программы). Приготовление цикла состоит в засылке нуля в ячейку 018 и в регистр M00. В цикле три блока. Однократное выполнение блоков MUL018017018 и ADD018000018 (последнего с модификацией) равносильно одному шагу по схеме Горнера. Сегмент C второго блока, равный адресу коэффициента a_0, модифицируется с помощью регистра M00, содержащего по очереди числа $0, 1, \ldots, 16$ (содержимое этого регистра увеличивается на единицу третьим блоком цикла, INA+001M00M00). Благодаря этому в вычислениях участвуют очередные коэффициенты $a_0, a_1, \ldots, a_{16}$.

Сравнение таким образом оформленных циклических программ с типичными внутренними кодами дает повод к следующим замечаниям. Во внешнем коде: 1° — изменение адресов команд заменено изменением содержания регистров, что более экономно, так как один регистр может обслуживать много сегментов, если они одинаково изменяются, 2° — не производится восстановления первоначального вида сегментов, ибо они в самой программе не меняются, 3° — нет проверки количества повторений цикла. Если программа содержит циклы высших порядков, то один сегмент может модифицироваться по-разному по отношению к циклам различных порядков. Тогда могут понадобиться даже три графы для обозначения модификации сегментов.

Помещенная в конце работы программа IV представляет собой пример более сложной программы. Она может предшествовать во времени программе III, так как по ней вычисляются коэффициенты полинома $\omega(x) = a_0x^{16}+a_1x^{15}+\ldots+a_{16}$, удовлетворяющего условиям $\omega(x_i) = y_i$ $(i = 0, 1, \ldots, 16)$. Числа $y_0, y_1, \ldots, y_{16}$ и $x_0, x_1, \ldots, x_{16}$ даны соответственно в ячейках 000, 001, ..., 016 и в ячейках 019, 020, ..., 035.

Первая часть программы соответствует вычислению коэффициентов b_i интерполяционной формулы Ньютона:

$$(3)\quad \omega(x) = b_0(x-x_1)(x-x_2)\ldots(x-x_{16})+b_1(x-x_2)\ldots(x-x_{16})+\ldots \\ \ldots+b_{15}(x-x_{16})+b_{16}.$$

Эти коэффициенты, равные разделенным разностям значений y_i, получаются на месте последних (b_0 в 000, b_1 в 001 и т. д.). Во второй части программы формула (3) переходит в формулу $\omega(x) = a_0x^{16}+ +a_1x^{15}+\ldots+a_{15}x+a_{16}$ при помощи последовательного свертывания слагаемых суммы (3). Коэффициенты $a_0, a_1, \ldots, a_{16}$ находятся после этого в ячейках 000, 001, ..., 016.

3. Программирующая и интерпретирующая программы. ПП и ИП, применяющиеся к внешнему коду, зависят от внутреннего кода цифровой машины и других ее параметров. Следовательно, мы можем дать лишь беглое описание этих программ.

ПП является по существу вводной программой. Непосредственно при вводе или после ввода программы во внешнем коде все внешние адреса (чисел, регистров модификации) и номера сегментов заменяются внутренними адресами. Подобному преобразованию подвергаются коды элементарных и обобщенных блоков, переходя в начальные адреса подпрограмм этих блоков. Для всего этого необходимо раньше определить взаимное расположение в памяти основных массивов — вводимой программы, чисел и регистров модификации, подпрограмм блоков, а также ПП и создаваемых ею вспомогательных величин. Для дальнейшей обработки с помощью ИП остаются только символы из граф от 2-й до 7-й. Роль номеров из 1-й графы сводится к определению внутренних адресов соответствующих сегментов — адресов, которые упоминаются в других местах программы.

Правильное функционирование ИП зависит от возможности определения типа любого сегмента или другого символа, составляющего часть программы во внешнем коде. Поэтому ПП выполняет еще одно задание: опираясь на внешний вид символа, ПП придает ему в памяти некоторую примету, взаимно однозначно связанную с типом этого символа.

Если машина — одноадресная, то любой символ одного из шести типов, определенных в § 2, помещается в ячейку длиной в одну команду. В случае трехадресной машины, когда ячейка содержит около 40 двоичных разрядов, можно объединить в одной ячейке символы целой строки бланка. При этом часть ячейки будет нередко неиспользованной, но ИП получится проще, чем в случае искусственного деления ячейки на „одноадресные" части.

ИП выполняет основную часть работы, связанной с обработкой программы во внешнем коде. ИП начинает действовать, когда в памяти машины находится уже программа, преобразованная с помощью ПП, и употребляемые ею подпрограммы (во внутреннем коде) всех элементарных блоков. В случае простейшей программы, не содержащей ни циклов, ни обобщенных блоков, один шаг работы ИП заключается в постановке модифицированных сегментов блока, который надо исполнить, на некоторые стандартные места и в обращении к подпрограмме этого блока. Подпрограмма должна отыскивать аргументы и адреса результатов с помощью этих стандартных мест.

Скобка „(", с которой начинается цикл, является сигналом к засылке кратности этого цикла в специальный счетчик, а адреса начала цикла — в некоторую вспомогательную ячейку. Скобка „)", которой кончается цикл, вызывает уменьшение на единицу содержимого соответствующего счетчика. Если полученный результат больше нуля, то следующим выполняется первый блок цикла, адрес которого ИП раньше запомнила в вышеупомянутой вспомогательной ячейке. В противном случае следующим выполняется блок, стоящий непосредственно за скобкой „)".

Если изучаемый блок — обобщенный, то это вызывает переход к выполнению элементарных блоков, из которых составлена программа этого обобщенного блока. Одновременно ИП запоминает адрес блока, следующего в программе за обобщенным блоком. Скобка „)", которой кончается подпрограмма обобщенного блока, является сигналом возвращения к основной программе, согласно запомненному адресу. Легко обнаружить сходство механизмов цикла и подпрограммы обобщенного блока, и использовать это сходство для сокращения ИП.

В заключение мы сделаем еще несколько замечаний.

I. Одно из основных достоинств внешнего кода, заключающееся в его сжатости, выявится во всей полноте лишь тогда, когда среди элементарных блоков будут более сложные, чем перечисленные в § 2. Вместе с тем, чем крупнее блоки, тем короче работа ИП по сравнению с самим счетом.

II. Если цифровая машина обладает достаточно емкой внешней памятью (например на магнитной ленте), то применение внешнего

кода особенно выгодно, так как подпрограммы всех элементарных блоков можно раз навсегда поместить в эту память.

III. Внешний код не устраняет необходимости распределения программы и числового материала в памяти, но облегчает эту задачу, ибо программа — безадресная, а для чисел и целых параметров введены независимые системы адресов.

IV. Не все факты, изложенные в § 2, одинаково существенны для описанного там внешнего кода. Для любого сегмента мы должны суметь однозначно определить его тип. Сохраняя это условие мы можем, например, ограничиться десятью регистрами модификации, увеличивая одновременно число номеров сегментов. Можно также в случае надобности ввести более сложные коды блоков. Конечно, подобные изменения в коде отражаются на программирующей и интерпретирующей программах.

ПРОГРАММА I

1	3
	EXP 000 002
	EQJ 000 001 S00
	SUB 000 001 003
	DIV 002 003 002
	SIN 003 003
	MUL 002 003 002
S00	STO

ПРОГРАММА II

1	2	3	4
	(	EXS	
		EXP VB VD	
		EQJ VB VC S00	
		SUB VB VC 003	
		DIV VD 003 VD	
		SIN 003 003	
		MUL VD 003 VD	
S00		STO	)

ПРОГРАММА III

2	3	4	5
	ZER 018		
	ZER M00		
(	+017		
	MUL 018 017 018		
	ADD 018 000 018		M00
	INA +001 M00 M00	)	
	STO		

ПРОГРАММА IV

Часть 1-я

2	3	4	5	6
	ZER M00			
	ZER M01			
(	+016			
	ZER M02			
(	+016		M00	
	SUB 001 000 000		 M02 M02 M02	
	SUB 020 019 036		 M02 M02	M01
	DIV 000 036 000		 M02 M02	
	INA +001 M02 M02	)		
	INA −001 M00 M00			
	INA +001 M01 M01	)		

Часть 2-я

2	3	4	5	6
	ZER M00			
(	+016			
	ZER M01			
(	+001		M00	
	MUL 000 020 036		 M01 M00	 M00
	SUB 001 036 001		 M01 M01	 M00 M00
	INA −001 M01 M01	)		
	INA +001 M00 M00	)		
	STO			

INSTYTUT MATEMATYCZNY POLSKIEJ AKADEMII NAUK
INSTYTUT BADAŃ JĄDROWYCH POLSKIEJ AKADEMII NAUK
МАТЕМАТИЧЕСКИЙ ИНСТИТУТ ПОЛЬСКОЙ АКАДЕМИИ НАУК
ИНСТИТУТ ЯДЕРНЫХ ИССЛЕДОВАНИЙ ПОЛЬСКОЙ АКАДЕМИИ НАУК

Praca wpłynęła 9. 1. 1960

S. PASZKOWSKI (Warszawa)

KOD ZEWNĘTRZNY DLA AUTOMATYCZNYCH MASZYN CYFROWYCH

STRESZCZENIE

W pracy opisano projekt kodu zewnętrznego, służącego do programowania obliczeń na automatycznych maszynach cyfrowych. Kod jest przeznaczony przede wszystkim dla niewielkich maszyn i dlatego jest dość prymitywny. Opracowując go autor nie usiłował upodobnić postaci programu w kodzie zewnętrznym do opisu programowanego zadania w powszechnie przyjętej symbolice matematycznej, lecz uwzględnił w tym kodzie specyficzne cechy typowych kodów wewnętrznych. Pozwoliło to znacznie skrócić programy, służące do interpretacji kodu zewnętrznego. Praca zawiera również ich opis.

Podstawowymi elementami opisywanego kodu są 1° bloki (odpowiednik rozkazów kodu wewnętrznego), składające się z jednego, dwóch lub większej liczby segmentów różnych typów (uogólnienie części operacyjnej i części adresowej rozkazu), 2° nawiasy i wskaźniki modyfikacji segmentów, służące do tworzenia cykli i podprogramów. Opis kodu jest zilustrowany 4 przykładami, m. in. programami obliczania współczynników i wartości wielomianu.

S. PASZKOWSKI (Warszawa)

EXTERNAL CODE FOR AUTOMATIC DIGITAL COMPUTERS

SUMMARY

The paper describes a project of an external code serving to programme calculations on automatic digital computers. The code is meant, above all, for small computers and consequently it is rather crude. Preparing it, the author did not endeavour to give the programme in the external code a form similar to a description of a programmed problem in the generally accepted mathematical notation, but he took into consideration in that code certain specific characteristics of typical internal codes. This made it possible to shorten considerably the routines used for the interpretation of the external code. The paper contains also their description.

The basic elements of the code in question are 1° blocks (corresponding to orders of the internal code) consisting of one, two of more segments of various types (a generalization of the operation part and the address part of an order), 2° brackets and segment modification indices used to form cycles and subroutines. The description of the code is illustrated by 4 examples, including the routines of calculating the coefficients and the values of a polynomial.
