RICE INSTITUTE COMPUTER PROJECT

AP1 - A Basic Assembly Program

January 1, 1959

1. Introduction.

The simplified system of coding for the Rice computer which is set out below is derived from three considerations: (i) the minimum machine configuration, which will be the first working form of the computer; (ii) the first version of the assembly program (AP1) which is designed to be consistent with (i); and (iii) the desirability of having an introductory system for teaching purposes. Accordingly, the role of certain short fast registers (PF2, Mode and trapping) and the use of tags are not discussed, nor is provision made for the synthesis of orders and use of correction tapes mentioned in the first full version of the assembly program (this is subsequently referred to as AP2; see Appendix 2 of the Computer Manual dated September 1st 1958). Whilst provision will be made in later versions of AP for more elaborate coding in symbolic form, so that the present system will be a sub-system of the first, certain inconsistencies have been noted in AP2 and these are corrected in AP1.

Although the system described below is complete in itself, it does not exclude the use of more elaborate programming, particularly in Field 2, by employing absolute octal codes. The user is referred to the full version of the Computer Manual for details. Symbolic operation codes are used in AP1 which, as far as possible, are consistent with current computer usage.

2. Machine Configuration.

(i) Storage. Octal addresses 00010 - 17777 are available for general (M) storage.

(ii) Arithmetic registers. All eight special registers (00000 to 00007) are available.

(iii) Control registers. All eight registers are available.

(iv) Input-Output. Most input and output will be by subroutine. Paper tape reader and punch are available,

together with line printer. No magnetic tape units are assumed.

(v) Special registers. A 15-bit sense light register is used (address 77771). Reference may be made in fields 3 and 4 to the X register. It is possible to enter the repeat mode from the program, and turn sense lights on and off by coding or manually, but for full use of the special registers in AP1 absolute code must be used.

3. Symbolic codes.

The general form of an instruction as it is punched on paper tape is as follows:

LOCNt SETUt OPNt ADDR MOD, AUXt REMARKScr

where "t" denotes the "tabulate" character, and "cr" denotes "carriage return". In correspondence with the absolute form of the instruction LOCN gives the symbolic form (if any) of the address in which it is found, SETU corresponds to Field 1, OPN to Field 2, AUX to Field 3, and ADDR + MOD to Field 4. Remarks are for the guidance of the coder only and are not read and reproduced by the computer unless preceded by a "REM" pseudo-order. Any field may be left blank, consistent with a meaningful construction being placed on the order. Any field may contain absolute octal codes, which will then be placed in the corresponding positions of the machine instruction, ignoring any bits which overflow to the left. The numerical portion of any field is assumed to be octal unless preceded immediately by the special symbol "d", meaning "decimal". The IM and IA bits are controlled by special inflexions on the ADDR + MOD field. Precise definitions of the allowed symbols are as follows:

Type (i) General storage address: Any upper case Roman letter followed by up to five upper case Roman letters or numerals. Examples: B, M3, COMMON, ZETA2. These symbols may only appear in the LOCN or ADDR fields. Up to 500 type (i) symbols may be used in AP1.

Type (ii) Special symbolic addresses: By convention we recognize the following symbols for "fast" addresses: Z, U, R, S, T4, T5, T6, T7 (A series); and CC, B1, B2, B3, B4, B5, B6, PF (B series). These may appear in SETU, ADDR+MOD, and AUX fields. If they appear in the LOCN field the order will be ignored. Use of the above symbols, and of X (for the X-register) should be restricted to the special significance associated with the Rice computer.

Type (iii) Special characters: *, a, d, + , -, |, →, (,), tab, cr, and , (comma).

Each field of the symbolic instruction has a well-defined form and if this is not recognised by the machine, a note is made on the printed listing of the program and an effective "NOP" (no operation) instruction is inserted. The acceptable symbols in each field are as follows:

(a) LOCN. May be blank or absolute or symbolic. If absolute, any octal address in the range 10 - 17777 is acceptable causing the location counter to be reset to this value; or a decimal address (preceded by the symbol "d") in the range 8 - 8191. If symbolic, any type (i) symbol may be used. An A series symbol causes the whole instruction to be ignored. A B series symbol sets up an error condition.

(b) SETU. May be blank or F, where F is a type (ii) symbolic address, or any of the forms -F, |F| or -|F| .

(c) OPN. May be blank or any absolute octal code, or one of the mnemonic orders or pseudo-orders defined in AP1. A symbolic operation is either a 3-letter mnemonic or, in the case of conditional transfers, it has the form IF(CCC)TTT where CCC is a test condition and TTT is a mnemonic for a transfer order.

Note: If (a), (b), and (c) are blank fields, the order is interpreted as a continuation of the preceding one (i.e., an overflow of the ADDR+ MOD, AUX or REMARKS fields).

(d) ADDR+ MOD. ADDR may be blank, absolute or symbolic in exactly the same way as LOCN. MOD is always one or more of the type (ii) B series symbols, connected to ADDR by + signs. In addition, this field may contain a "relative" part consisting of an octal or decimal integer preceded by a + or - sign. The IA bit is controlled by the * symbol, which sets IA=1 when it appears in this field. If M is an allowed ADDR MOD symbol in the above sense then so are -M, |M| and -|M| , where the special characters control IM bits 2 and 3. IM bit 1 is set to zero except in the following cases:

(i) The symbol "a" appears in this field.

(ii) The STO, SLF, SLN, TRA, or HTR orders are used.

(iii) ADDR is blank but B-modifiers appear in MOD.

(iv) OPN is class 4 and ADDR is absolute.

Examples of valid symbols in this field are:

```
COMMON + 1
*ZETA
-|A| B1-2
d48
-ad122 + B1
B5
```

Any symbols appearing in parentheses in this field are treated as the Z character by API, so that an address which is modified by the program may be annotated conveniently, e.g., (FWA) + B1 + B2 is treated as Z + B1 + B2.

(e) AUX. This may be blank or one of the forms U→F, R→F, I Bi, Bi + 1, Bi-1, or Bi + X, where Bi stands for one of the special symbols, type (ii) B series, F is any type (ii) symbol, and I refers to the M portion of the I register.

(f) REMARKS. This field is ignored by the computer, as indicated above, unless a REM pseudo-order is used, in which case all symbols in the Flexowriter code which may be translated into printer code are reproduced in the printed listing. Any symbol which cannot be translated is left blank.

4. The process of assembly.

Standard coding sheets are provided for the preparation of programs. Provided the conventions of the previous section are observed in transcribing the program to paper tape there will be no ambiguity in the code which is presented to the computer. AP1 assumes only 4089 words of general storage although programs using more storage may be assembled without difficulty. There are two distinct phases of assembly, each requiring the punched paper tape to be read. In the first pass, each field is recognised and a symbol table prepared. An on-line listing may be obtained of unassembled orders during this stage. In case of any fields invalid in the sense described above a printed query is raised. At the end of phase 1, the symbol table is held in storage, but may be obtained in punched paper tape form.

Phase 2 is initiated either immediately following phase 1 or as a separate process, and assigns the final form to each instruction. Small programs may be held in storage until the execution stage. Otherwise, a binary tape will be punched and at the same time, a printed final version of the program may be obtained.

5. Subroutine library.

AP1 contains an index of library subroutine names, working storage, fast stores used, and space required. Any unassigned symbol is compared at the end of phase 2 with entries in this table and if it appears there a request is made for the subroutine, which is drawn from the library manually and read into the machine.

By convention, all library subroutines are written in completely relative form, and it is only necessary to know the address at which they are to start in order to relocate them. From the abstract of the program the coder must determine which fast registers are to be saved and make provision for doing so. In general, subroutines are written to use B-registers in the order B6, B5 ... as required, to use T4...T7 for arguments and results, and to provide separate error returns to various points in the calling sequence. There is no "PACK" type instruction in AP1. Transfer to subroutines is by an unconditional TRA order. Library subroutines generally provide their own working storage (if this is only one or two cells) or request the coder to specify a working storage region as a parameter in the calling sequence.

6. Types of order.

A full summary of the operations available in the AP1 is given at the end of this guide, and exact details of their function are given in the Computer Manual. The following remarks generally apply when AP1 is used.

Class 0: On both conditional and unconditional TRA and HTR orders IM bit 1 is set to 1 to avoid a fetch time which is otherwise wasted. When B registers are compared using the SKP order, an order such as

```
B1    IF(NEG)SKP   B2     is recognised as
B1    IF(NEG)SKP   aZ + B2
```

whereas if we wish to compare B1 with ((B2)) this would be written as B1 IF(NEG)SKP Z + B2. This is a consequence of the rule of formation given in paragraph 3 (d)(iii) and affects both numerical and logical work using the contents of B-registers.

Class 1: -

Class 2: Two conventional store mnemonics are provided (FST and STO) differing only in the IM bit, which is set to 0 or 1 respectively.

Class 4: Since shifts and B-modifying orders most frequently use the M field as an absolute quantity (andin any case S is not used in this class) a fetch time can normally be avoided by setting IM bit 1= 1 and this is done automatically for absolute octal or decimal ADDR fields which are not indirectly addressed. It can, of course, be controlled on other orders by the "a" symbol. In B-registers 1's complement arithmetic is used, so that a negative ADDR field appearing with SB1 or AB1 type orders is first complemented. However, IM bit 3 is also set to 1 as a guide to the assembly program; e.g., SB2-23 is assembled as 0044200005000077754.

Only the most common shifts are provided in AP1. Complex logical shifts must be coded in absolute form.

In setting and testing sense lights, ADDR gives in octal form the exact bit pattern corresponding to the lights being used. Thus the order

SLN 71

turns on sense lights 10, 11, 12, and 15 and sets IM bit 1 to 1.

Class 5: The function of complementing after a logical operation is achieved by placing a "-" sign before one of the four basic operations.

Class 6:

Class 7: No special functions are contemplated initially, apart from NOP(70000). Note that the full "no-operation" order is

U NOP S

Summary of API Mnemonics

Mnemonic Code	Octal Code*	Name of Order	Remarks
IF(CCC)HTR	00aaa	Conditional halt and transfer	Transfer takes
IF(CCC)TRA	01aaa	Conditional transfer	place if any
IF(CCC)SKP	02aaa	Conditional skip (by 1 instruction)	CCC satisfied.
IF(CCC)JMP	03aaa	Conditional jump by (x)	
HTR	00000	Unconditional halt and transfer	
TRA	01000	Unconditional transfer	
SKP	02000	Unconditional skip	
JMP	03000	Unconditional jump by (x)	
POS	0011a	Mantissa positive or zero	≥ 0
PNZ	0415a	Mantissa positive and not zero	> 0
NEG	0051a	Mantissa negative or zero	≤ 0
NNZ	0455a	Mantissa negative and not zero	< 0
MOV	0a2aa	Mantissa overflow indicator on	
NMO	0a6aa	Mantissa overflow indicator off	
EOV	0a3aa	Exponent overflow indicator on	
NEO	0a7aa	Exponent overflow indicator off	
ZER	0aa1a	Mantissa zero	$= 0$
NZE	0aa5a	Mantissa non-zero	$\neq 0$
EVN	0aa2a	Bit 54 = 0	
ODD	0aa6a	Bit 54 = 1	
SLN	0aa3a	Test for sense lights on	
SLF	0aa7a	Test for sense lights off	
NUL	0aa4a	Test for all 54 bits zero	
CLA	0050nnn	Clear and add to U	Field 1 set to 00
ADD	1n0nn	Fixed point add	
SUB	1n1nn	Fixed point subtract	
MPY	1n2nn	Fixed point multiply	
DIV	1n3nn	Fixed point divide	

*In the octal codes, n indicates that a digit position is not available for synthesizing other orders, a indicates that it is available. Unless other specifications are made, all these digits are set to zero.

Mnemonic Code	Octal Code	Name of Order	Remarks
FAD	1n4nn	Floating point add	
FSB	1n5nn	Floating point subtract	
FMP	1n6nn	Floating point multiply	
FDV	1n7nn	Floating point divide	
STO	20nna	Store	IM bit 1 = 1
FST	20nna	Fetch and store	IM bit 1 = 0
STS	22nna	Store sum	
RPM	21n0a	Replace M digits	
RPR	21n1a	Replace right half word	
RPL	21n2a	Replace left half word	
ACC	400nn	Add to Control Counter	
AB1...etc.	401nn...	Add to B1...etc.	
APF	407nn	Add to pathfinder	
SCC	440nn	Set Control Counter	
SB1...etc.	441nn...	Set B1...etc.	
SPF	447nn	Set pathfinder	
UMR	41n1a	U mantissa right	
UML	41n2a	U mantissa left	
RMR	41na1	R mantissa right	
RML	41na2	R mantissa left	
DMR	41n15	Double mantissa right	
DML	41n62	Double mantissa left	
LUR	45n1a	Logical U right	
LUL	45n2a	Logical U left	
LRR	45na1	Logical R right	
LRL	45na2	Logical R left	
LRS	45n15	Long right shift	
LLS	45n62	Long left shift	

Mnemonic Code	Octal Code	Name of Order	Remarks
BCT	43n01	Bit count	
SLN	42n0n	Sense lights on	
SLF	42n1n	Sense lights off	
ERM	46n1n...	(M) Enter Repeat mode	ADDR + MOD field cannot be used.
AND	50nnn	And to U	A preceding
ORU	51nnn	Or to U	"-" sign com-
SYM	52nnn	Symmetric difference to U	plements the
XTR	53nnn	Extract S through R to U	result
PRk	62nkn	Print with format k (here k = 0,1,2...7)	
RHX	600nn	Read hexads	
PHX	604nn	Punch hexads	
NOP	70000	No operation	

Pseudo-Orders in APl

ORG — Origin. The ADDR field gives the location at which the following instruction is to be placed.

BSS — Block started by symbol. The symbol in the LOCN field is assigned the current value of the Location counter plus one. The quantity in the ADDR field must be an absolute integer or some previously assigned symbol. This quantity (or its equivalent) minus one is added to the Location counter before proceeding.

BES — Block ended by symbol. This pseudo-order is similar to BSS except that the Location counter is advanced by the quantity in the ADDR field before the symbol in the LOCN field is assigned.

DEC — Decimal data. The numbers which follow are recognized as floating or fixed point decimal quantities and stored in consecutive memory locations starting with the current value of the location counter. Numbers are separated either by a "cr" symbol or a ",". Floating point numbers contain a decimal point in the mantissa and any number may contain a signed integral exponent preceded by the letter E. Example:

```
    1.0          Stored as normalised
    347.5 E9      floating point numbers

    1            Stored as
   -3E7           fixed point
    97             integers.
```

OCT — Octal data. The numbers which follow are recognized as fixed point octal integers and stored in consecutive memory locations starting with the current value of the location counter.

For any form of data input, the range of magnitude of floating point numbers F is given by

$$(256)^{-31} \cdot (2^{-7}) \leqslant |F| \leqslant (256)^{31} \quad (1-2^{-47})$$

which is approximately

$$10^{-77} \leqslant |F| \leqslant 10^{74} \quad .$$

All floating point numbers read into the machine are normalised. For fixed point integers X, we must have

$$0 \leqslant |X| \leqslant 2^{54}$$

Any negative number is evaluated by first obtaining the positive binary magnitude and then complementing this. If $-2^{48} < X \leqslant -1$ and X is an integer, the exponent is not complemented, but set to zero.

BCD — Binary coded data. All characters which follow this order are stored nine to a word from the current location onwards, in succession from left to right, in Flexowriter code. The sequence is ended by a double "cr". Any spaces unused in the last word are cleared to zero.

END — End of symbolic program. Terminating routines of AP1 Phase 1 or 2 are initiated by this command.

EQU — Equivalence. The type (i) symbol in the LOCN field is given the equivalent currently assigned to the symbol in the ADDR field. Once defined, an equivalent cannot be altered. It is not necessary to distinguish between preset and program parameters.

REM — Remarks. All characters and symbols following this order are reproduced on the printed program listing without affecting the assembly process. The "Remark" is terminated by a double "cr". The LOCN and SETU fields are ignored.

Examples of AP1 Coding

1. Multiplication of two complex numbers in floating point form.

We have $(x_1 + ix_2)(y_1 + iy_2) = z_1 + iz_2 = x_1y_1 - x_2y_2 + i(x_1y_2 + x_2y_1)$

where (X1) = x_1, (X2) = x_2. . . etc.

A method of coding this would be:

```
CMPY        CLA   X1, U→T4
            FMP   Y1, U→T5
            CLA   X2, U→T6
            FMP  -Y2
            FAD   T5
            STO   Z1
         T4 FMP   Y2, U→T5
         T6 FMP   Y1
            FAD   T5
            STO   Z2
```

2. Polynomial evaluation.

$$Y = \sum_{i=0}^{k} a_i t^i$$

where (TLOCN) = t, (A) = a_0, (A + 1) = a_1, ... (A + i) = a_i, ...

(A + K) = a_K and $(KLOCN)_M$ = K.

A code for this calculation is:

```
POLY    Z   SB1  *KLOCN, U→T4
            CLA   TLOCN, U→T5
 AB     T4  FMP   T5
            FAD   A+B1, U→T4
        B1  IF(NZE)TRA  AB, B1 - 1
        T4  STO   Y
```