CGTM 96
Howard Cohen
Maurice Schlumberger
September 1970

USER GUIDE FOR THE IOP 7000 MINIFLOW ASSEMBLER AND IOP SIMULATOR

TABLE OF CONTENTS

Input Source

Input is expected as card image data, one assembler source statement per card record - no continuation. Only columns 1-72 are scanned even though the entire 80 columns is printed. For submission of a Job see the section on JCL.

Control Cards

There is, at present, one control card which is optional and must precede the source code. The format is:

(i) '$$' in columns 1-2

(ii) The 4 character control word in columns 3-6

The existing control is:

LIST - give a source code listing including error diagnostics, object code, and cross reference table.

Format of Source Statement

* comment[1]

[LABEL] op-code [operands][2] [comments]

Labels: Labels may be up to 10 characters in length. They must begin -- see JCL, Assembler Execution (page 7) -- with an alphabetic character and must start in the first column. Otherwise all characters except '+', '-', '*', ',', and ' ' are permitted (any instance of the above characters is an error and implies the end of scanning for the label).

[1] comment card begins with an asterisk (asterisk may be in any column).

[2] [operands] depends on the op-code.

* Reference - ICAP II, form 801027 and Principles of Operation, form 807003-2 .

Op codes: If there is a label then there must be at least one blank between the label and the assembler instruction. Otherwise the mnemonic instruction may begin in the second column or beyond. The actual codes are identical with those in the Principles of Operation manual except for the following additions:

OCH	defines, like OCT, an octal constant, but only of length 18 bits (half-word)
TRACE	at simulation time gives a display of the IOP (excluding channel registers or control memory.)
NOTR	halts the trace initiated by TRACE
DUMP a,b	dumps control memory and main memory: 'a' and 'b' designate (in units of 1/4 K words) the amount of control and main memory, respectively, to be dumped (where either or both operands are optional, having a default=0).

Flags: The flags to set bits in the assembler instructions must immediately follow the operation code (no blanks). A blank will halt the search for flags. The flags and their meanings for each instruction class are given below:

1. Engine class
   - ! - exit bit set[1]
   - ? - link bit set
   - · - return bit set
2. Single operand class[2]
   - * - use shift counter as it is - no operand will be scanned. (invalid for the COMplement instruction)
3. Transfer class
   - * - indirect (scratch memory)
   - ! - Transfer or exit (TY=11)[1]
   - · - Transfer or return (TY=10)
4. Data and control class
   - ! - exit bit set[1] except for channel
   - · - return bit set memory direct commands
   - * - indirect - for RIOP, WIOP, RGEN, WGEN, only - access IOP memory indirect, or access general register indirect

[1] An apostrophe may be used in place of the exclamation mark; in either instance the apostrophe will always be printed.

[2] If a ! is erroneously used, it will not appear on a listing -- be careful!

Notes on the Data Class instructions -- Description of Read Main Memory Indirect (refer to IOP manual - form B07003-2, IC7000 CPU Model 2,3, or Principles of Operation, form 807003-2).

RM and WM	BR=00	(AC1 and AC2)
RMDIR	BR=11	(AC1,AC2, and Directive Register)
RMDN	BR=10	(AC1,AC2, and Directive Register 0-6, 14-17)
RMWD,RMIND	BR=01	(AC1,AC2, and Directive Register 7-13, 18-35)

Operands

A missing operand will produce an error message. Following is a summary of the operand requirements for each instruction class (and sub-classes when necessary). Only Immediate operands can be negative. Blanks are not permitted in the operand field. Otherwise an operand may be any of the following:

1. the location counter (*), a full word address
2. a label
3. a number (octal or decimal) - cannot exceed +32767 or be less than -32768 (implementation restriction) except on OCT, OCH, and VFD octal operand.
4. an expression involving (1), (2), and (3) above and '+', '-', or '*', except only one forward reference is allowed and the forward reference, if X, cannot appear as -X, or *X.

Engine Class

(i)	immediate instructions[1]	A, I
(ii)	scratch memory instructions (except)	A, E
(iii)	CLR, ZERO, LD1	A

[1] The pseudo immediate instructions (CM, SM, etc.) which require a two's complement of the second operand are also A, I.

Single Operand Class		
(i)	COM	A
(ii)	SCT*	none
(iii)	SCT,ROT	S
(iv)	(SRT, SLT)	A, S
(v)	(SRT*, SLT*, ROT*)	A
Transfer Class		
(i)	(EXIT, NOP, RET)	none
(ii)	direct	T
(iii)	indirect	E
Control Class		none

Data Class

(i)	RIOP, WIOP	G
(ii)	RIOP*, WIOP*	E
(iii)	(RM, RMIND, RMDN, RMDIR, WM, RMDW)	E
(iv)	STO	A, E
(v)	RGEN, WGEN	F
(vi)	RGEN*, WGEN*	E

Explanation of symbols

A - accumulator specification (bits 3-4)

I - immediate operand (bits 6-17)

E - scratch memory operand (bits 13-17)

S - shift count specification (bits 11-17)

T - direct transfer address (bits 8-17)

G - IOP memory address (bits 8-17)

F - general register (bits 14-17)

Overflow

Overflow is checked for in all the above fields. This produces the error diagnostic 'OS'.

Pseudo commands

ORG, BSS, EQU -- a single operand, not forward referenced

OCT, OCH -- a numeric character string of precisely 12 and 6 characters, respectively.

VFD -- a minimum of one bit specification (e.g. 36/value). Any number of bit specifications per command of which at most two are forward referenced (see (5) on page 6).

Examples

(i)	ORG	XXXX	XXXX is a decimal number less than 1024
	ORG	0/XXXX	XXXX is an octal number less than 1777
	ORG	Label	label must be defined
	ORG	expression	variables may be any of the above three. '+', '-', and '*' are the only valid operators.
(ii)	BSS	<same as above>	reserves a number of words equal to the value of the operand
(iii)	EQU	<same as above>	sets the label of this source card to the value specified by the operand
(iv)	OCT	XXXXXXXXXXXX	12 octal characters - stored in a full word
	OCH	XXXXXX	6 octal characters - stored in a half word. A check is made for a digit greater than 7 on both OCT and OCH.

(v) VFD $X_1/E_1,X_2/E_2,\ldots,X_n/E_n$ where $X_1,\ldots, X_n$ may be either

(a) a positive decimal number, or

(b) a decimal number preceded by the letter '0' which specifies that the corresponding E_i is to be considered octal.

$X_1+X_2+ \ldots + X_n$ must be <u>exactly</u> equal to 36.

$E_1, E_2, \ldots E_n$ may have any of the following forms if the corresponding X_i is not preceded by the letter '0':

(1) a numeric character string

(2) a label

(3) an expression involving either or both of the above.

Only two E_i's are allowed to have forward references and only one forward reference per each of these two E_i's. If the

corresponding X_i is preceded by the letter '0', then E_i must be an octal numeric string (no labels or expressions).

(c) Note: If X_i specifies octal (preceded by the letter 0) then it may be $\alpha = 36$; otherwise, $X_i > 15$ will produce a warning message since an implementation restriction will produce a truncation to 15 bits.

Octal Representation

A number (or E field in the VFD command) is made octal by preceding it with '0/' (e.g. 036/XXX for the VFD command).

The syntax of the operand field is similar to that in the ICAP Manual except possibly for the following:

(1) Blanks will terminate an operand field; it must be packed.

(2) Expressions are allowed (addition, multiplication, and subtraction)

(3) Forward references are allowed, but only one per operand. Suppose 'X' is a forward reference: 'X*N' is not allowed if 'X' is a forward reference, but 'X-N' or 'X+N' is allowed. ('-X' is not)

(4) Forward referencing in the VFD command is permitted, but only two forward references within the 36 bit word. Expressions are allowed in the VFD operands (exception, see (5)).

(5) The VFD operand before a '/' must be numeric (except for an optional preceding 'Ø'). If the 'Ø' is there, then the field following the slash must be numeric - no expressions are permitted in this case.

(6) At least one blank between comments and operand fields

(7) In VFD, if the slash operand is not octal, then only 15 bits of significance are kept.

JCL - Assembler Execution

To execute an assembly and/or simulation do the following:

(1) Under CRBE

(i) Create and save an FLIST file as follows (with your own JOB card in place of line 100):

```
 100 //HDCXXXXX JOB 'HDC$CG',039,L=10K,CLASS=E,REGION=300K,GRAPH=YES
 200 //STEP1  EXEC PLLOADGO
 300 //GO.SYSLIB  DD  DSNAME=USER.HES.V5PL1LIB
 400 //GO.SYSLIN  DD  DSNAME=PUB.JEG.LIBRARYP(COHEMA),UNIT=2314,
 500 //         VOL=SER=PUB003,DISP=(OLD,KEEP)
 700 //GO.SOURCE  DD  DSNAME=&&STEMP,DISP=NEW,UNIT=SYSDA,
 800 //         DCB=(RECFM=FB,LRECL=132,BLKSIZE=3432),SPACE=(TRK,(10,1))
 900 //GO.COREMAP DD  DSNAME=&&COREMAP,UNIT=SYSDA,SPACE=(TRK,(5,1)),
1000 //         DISP=(NEW,PASS),DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
1150 //GO.SYSIN  DD  *
1175 =*
1180 /*
1200 //STEP2  EXEC PLLOADGO,COND=(8,LE,STEP1.GO)
1300 //GO.SYSLIB  DD  DSNAME=USER.HES.V5PL1LIB
1400 //GO.SYSLIN  DD  DSNAME=PUB.JEG.OVPCCALC(SCHLUM1),UNIT=2314,
1500 //         VOL=SER=PUB002,DISP=(OLD,KEEP)
1900 //GO.MAURICE DD  DSNAME=*.STEP1.GO.COREMAP,DISP=(OLD,PASS)
2200 //GO.HOWARD DD  DSNAME=PUB.JEG.LIBRARYP(SCHLUM2),UNIT=2314,
2300 //         DISP=(OLD,KEEP),DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
2400 //         VOL=SER=PUB003
   %
```

STEP1 refers to the assembler GO step, and STEP2 refers to the interpreter GO step. On STEP2, 'COND=(8,LT,STEP1.GO)' is used to allow execution of the interpreter only if the error severity is 4 (warning) or less. PARM ='/DEBUG' on the STEP2 statement will invoke a limited trace by the simulator (see page 17).

(ii) Fetch, as your active file, the assembler source code, including control cards.

(iii) SUBMIT the file created under (i) above (takes the active file as the input source).

(2) Under WYLBUR (suggestion)

(i) Create and save three files:

(a) #IOPASM1 - line numbers 100-1150 above

(b) #IOPASM2 - line numbers 1180-2400 above

(c) SOURCE - your assembler source code

(ii) Create and save a fourth file consisting of the lines:

```
USE #IOPASM1 ON WYLXXX
COPY ALL TO END FROM SOURCE ON WYLXXX
COPY ALL TO END FROM #IOPASM2 ON WYLXXX
RUN
```

(iii) If the file in (ii) is called #DOIOP and resides on WYLXXX, then type: EXEC FROM #DOIOP ON WYLXXX

(3) In your file(s) there are four GO. cards

(i) GO.SOURCE specifies a temporary data set on which to write the source records, a record oriented file (see PL/I).

(ii) GO.COREMAP is a stream oriented file on which a map of control memory is written (a continuous stream of 36 bit words).

(iii) GO.MAURICE is the same file as GO.COREMAP; it is the source of the object code for the interpreter.

(iv) GO.HOWARD is a map of main memory in stream format.

(4) Permanent Object Module

To create a permanent object module, change the DSNAME on //GO.COREMAP to be a permanent data set (or member). Change DISP to '(OLD,KEEP)' and UNIT to '2314'. Specify VOL=SER=XXX. Then to use this permanent object code, run STEP2 described above, with DISP on the GO.MAURICE DD statement changed to '(OLD,KEEP)'.

OUTPUT ($$LIST specified as a control card)

If any control cards are included, they will be listed on the first page. Any terminal errors will appear on a second page.

There are two sections of OUTPUT; the source listing (and code), and the cross reference table.

(1) SOURCE Listing

(a) LOCATION - first column

(i) this specifies the octal address of the instruction or pseudo instruction on the source record, unless

(ii) the pseudo code is EQU, in which case the value listed is the octal value of the equate

(iii) comment cards have this column left blank.

(b) INSTRUCTION - second column

(i) The octal representation of the code generated for the two 18 bit instructions, or

(ii) EQU, BSS, and ORG do not generate anything for this field.

(c) SEQ - third column

This is a card counter and is used in the cross reference table.

(d) ERR - fourth column

The error code, if any, is generated by the source statement. '*' means more than one error was found, and '$' means

(i) a NOP was inserted in the second half of the previous instruction because the current statement must begin on a fullword boundary, or

(ii) a NOP was inserted in the second half of the current instruction because the instruction in the first half is CALL.

(e) The remainder of the print line is simply the card image.

(2) Cross Reference Table

(i) Symbols (or labels) are listed in alphabetical order in column two (headed by SYMBOL).

(ii) Column one - DEFN specifies the card sequence number at which the symbol was originally defined.

(iii) Column three - VALUE specifies the value (in octal) of the symbol (usually) the location counter at which it is defined - exception, EQU).

(iv) Column four - ADDRESS specifies descending list of card sequence numbers on which the symbol appears.

(3) Final Lines

There are three final lines. These lines are printed even if the $$LIST control card is omitted. The first prints the number of errors found in the program. This will not necessarily agree with the number of error codes found under ERR, since, at present, error recovery is made on warnings (severity=4), and at most only one error diagnostic can be printed for each source statement; thus, multiple errors are counted but not shown if greater than two. The second line prints the maximum error severity encountered, and the third line prints the number of times a NOP was inserted due to a fullword boundary alignment (if no such instances, then this line is omitted).

System Errors

If a system error occurs, an on-unit will list the 'on-code'. (see PL/I(F)V5 reference manual, page 310). The current source statement will be listed. Under it will be a vertical bar showing the current position of the scan pointer. Then the assembler will "pretend" that an end of file has been reached (e.g., a missing 'END' statement).

Terminal Errors

The only terminal errors at present occur when

(i) the location (or instruction) counter exceeds 1023, or

(ii) the number of input source statements exceeds 3000, or

(iii) symbol table overflow occurs.

-Case (i) is an implementation-free error since control memory is 1K. If the error occurs on a ORG or BSS, the instruction is ignored*. The error diagnostic 'OL' appears, and a severity of 12 is issued (this allows further processing). Otherwise, if any other instruction forces the program to exceed control memory capacity,

(a) the 'OL' diagnostic appears on the card

(b) the first page will contain a line indicating the particular terminal error involved

(c) the severity will be equal to 16

(d) Endfile of SYSIN is signalled.

-Case (ii) is not implementation-free. An arbitrary (meaning it can be changed) limit of 3000 input statements is built into the program. The CCLASS array dimension must be enlarged and the error check statement changed. This also produces a message on the first page, indicating this particular terminal error.

-Case (iii) is also implementation dependent - a maximum of 512 symbols is permitted. This can also be changed (see the documentation on the internals of the assembler).

Completion Code

The return code issued at the completion of the assembler GO STEP is either 0,4,8,12, or 16; i.e., the highest severity error encountered during compilation. This allows you to control the circumstances under which you desire the simulator to execute the object code generated (see JCL).

* However, BSS and ORG automatically force boundary alignment. This in itself may produce the terminal error even though the illegal operand has been ignored.

Error Messages

(1) Severity - All error messages have a severity. The highest severity found is given as the return code from STEP1. Execution of STEP2 is governed by the setting of the COND parameter (see 360/JCL Manual). The severities are shown in the table below:

SEV	Meaning
4	warning
8	error
12	severe error
16	terminal error (see below)

(2) Error messages occur on the output listing in the column entitled ERR (see Output section). An error immediately terminates code generation for that source card, unless it is only a warning message (see next page). Multiple errors on one source card cannot be shown simultaneously, but a '*' indicates that more than one was found. In most instances, a HALT is inserted as the code is generated. If the error occurs in or beyond the operands field, then one or two HALTS's, respectively, will be generated depending on whether the operation code specifies an 18 or 36 bit word. Finally, the 'undefined label error', 'UL', is also shown in the cross reference table under the VALUE column by inserting the word 'UNDEF'.

(3) Exception - If an undefined label operand occurs on either an EQU or an ORG, then the 'UL' diagnostic will not appear, but the 'NF' message will be listed. Not even a '*' will appear in place of 'UL'.

SEV	CODE		MEANING
8	CO		Operand constant magnitude (> 32767 or < -32768)
8	DP		Each operand of the DUMP statement must be numeric and less than or equal to three and greater than or equal to zero.
8	IB		In VFD command - the number of bits specified exceeds 36
12	IC		Invalid op code
4	IF		Illegal flag(s) following the op-code
12	IL		slash(es) in label operand or 1st character is not alphameric
8	IN		Invalid numerical field
12	IO		Invalid operand
		(1)	first char of label operand is not alphabetic
		(2)	* precedes expression
		(3)	'*' left on stack
		(4)	syntactical error
12	IS		Invalid label (> 10 characters)
8	LB		Label must be followed by blank
12	LL		Last column reached while scanning operand to right of '/' in VFD command
12	MD		Multiply defined label
12	MO	(1)	Last column is reached while still scanning operand,
		(2)	or missing operand
12	NF		The operand of an EQU or ORG (Origin) or BSS (Block starting symbol) cannot be forward referenced
12	NG		negative operand
4	NL		In an EQU (Equate) there should be a label

SEV	CODE		MEANING
12	NR		Too many forward references in a single operand
12	NO[1]	(1)	More than 5 operands in an expression
		(2)	More than 4 operators in an expression
12	OL		The location counter is greater than 1023. ORG and BSS are ignored
8	ON		OCT operand is not numeric or has a digit > 7
12	OP[1]		Not a valid operator (must be '+', '*', or '-')
8	OS		Operand overflow (trucation error) - warning in the case of a VFD operand (severity=4)
8	TO		Comma follows the 2nd operand
12	UL		Undefined label
12	VF		Field in VFD operand is greater than 10 characters long
4	VN		Number to left of slash in VFD command is greater than fifteen, or equal to zero
12	VO		Operand to right of slash in VFD command ('E' operand) is not numeric
12	VR		Too many forward references in VFD command (more than 2)

[1] These error messages may never occur (I can find no examples), but they remain in case such situations could occur, thus preventing a system error.

IOP INTERPRETER

The IOP Interpreter program reproduces, as far as possible, the actual IOP. (Form 807003-2)

The program is composed of 3 main parts:

Scheduler

Decoder

Service Routines: Display
Dump
Error

The object code is run using the SLAC WYLBUR file

MLS.CG.LIB(IOP) on WYL003, which has the following JCL for Release 18, MVT, and HASP. The PLI(F) version 5 library must be used.

```
//      JOB 'MLS$CG,(SCHLUMBERGER)',057,PRTY=8,CLASS=E,REGION=150K
//  EXEC PLLOADGO, PARM='/DEBUG'
//GO.SYSLIB DD DSN=USER.HES.V5PL1LIB
//    DD
//GO.SYSLIN DD DSN=PUB.JEG.OVPCCALC(SCHLUM1),DISP=(OLD,KEEP),
//      UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),VOL=SER=PUB002
//GO.MAURICE DD DSN=PUB.JEG.OVPCCALC(SCHLUM3),DISP=(OLD,KEEP),
//      UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),VOL=SER=PUB002
//GO.HOWARD DD DSN=PUB.JEG.LIBRARYP(SCHLUM2),DISP=(OLD,KEEP),
//      UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),VOL=SER=PUB003
```

The source program is, along with its JCL for compilation, in the SLAC WYLBUR file MLS.CG.PL1FUN on WYL001.

Currently, the simulator needs 150K for running and runs up to 500 miniflow instructions per second.

Scheduler

Does the "wired-in-sequencer" part of the job, except for a few features:

manual switch

reset stop

Furthermore, no display register has been implemented; we don't have any interpreted lights.

Decoder

Decodes and simulates instructions, as specified in the IOP Manual (Form 807003-2).

There are a few exceptions which we hope won't interfere with simulation: In the control class operand, bits 15, 16, 17 are used for decoding, with the following meanings (in octal)

0	Halt IOP
1	Unhang CPU (sets a flag for the MLP)
2	Set CPU trap
3	Noop (+ warning)
4	Set trace: gives a "display" at the end of each instruction
5	No trace, stops the previous
6	Dump: dumps the current state of the IOP, including channel registers and more or less of control and main memory, depending on the operands.

For Rotate AC and Normalize, we have ignored bit 4 in the decoding.

When normalizing, we issue a warning message if the AC is zero, and produce a Noop.

If, in the engine class, you try to manipulate all 3 AC's with something else than a Pass, a warning message will be issued, and all 3 AC's will be or-ed to form the AC operand before the specified operation is executed.

Data Access Class - Nothing is set for reading CPU control memory direct: DXUS should eventually do the job. So, we warn you and issue a Noop.

Error and Warning Messages

- *TILT*	Too many errors, stops you, set presently at 10 errors or warnings, prevents you from some bad and useless loops, and gives you a general dump.
- No 3 ACs	Warning
-Zero Value Normalize	Warning
-Illegal instruction	Displays, stop
-Out of bounds	Displays, ignores the command (call or return), goes to the next one sequentially.
-Illegal operation on channels	Dumps channel registers, stops.
-Scratch memory	Displays, Noop
-SM5	Dumps, stops
-Cannot yet interpret	Displays, Noop

Restrictions

-1K of main core, (statement #3 in IOP) but both main and control core are writable through the PL/1 files named HOWARD (main) and MAURICE (control). These have as DD cards:

```
//GO.MAURICE DD DSN=PUB.JEG.OVPCCALC(SCHLUM3),DISP=(OLD,KEEP),
//      UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),VOL=SER=PUB002
//GO.HOWARD DD DSN=PUB.JEG.LIBRARYP(SCHLUM2),DISP=(OLD,KEEP),
//      UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),VOL=SER=PUB003
```

-Both files HOWARD and MAURICE are PL/I STREAM, LIST files containing only coremaps in sequential order. See Appendix for example of a program which loads HOWARD.

-Stops after 10 messages.

-The channels are considered as 4 independent entities, the registers of which you read or write, but these channels are not yet either real or simulated, so you cannot get much out of them.

-There is a limit on the number of interpreted instructions which is presently set at 1500.

Output

-Through TRACE and DUMP commands, and at the end, you can have a good idea of what goes on.

-Right now the IOP is simulated just be itself without being linked to the MLP and real channels. As long as it doesn't have any I/O through channels, you cannot utilize true "output".

A few details:

-There are a few debugging features left in the program that you can set by indicating PARM = '/DEBUG' on the EXEC card in the WYL.CG.MLS.LIB(IOP) file. It gives the code of instructions interpreted and whatever is stored to main and control core.

```
/*      MEMORY RESET PROGRAM          */
RESET: PROCEDURE    OPTIONS(MAIN);
 DECLARE HOWARD  FILE STREAM OUTPUT;
 DECLARE MEMORY(0:1023) INITIAL ((1024) 0) BIT(36);

  MEMORY(0)='011010100000000000000000000000000000'B;
  MEMORY(1)='101000100000000001000000000000000000'B;
  MEMORY(2)='010111100000000001000000000000011001'B;
  MEMORY(3)='000001000000000010000000000000010110'B;
  MEMORY(4)='010000000000010010000000000000011010'B;
  MEMORY(5)='110000100000000000000011000000000010'B;
  MEMORY(6)='001000100000000000010100000000000010'B;
  MEMORY(7)='111000000000000000000110000000000011'B;
  MEMORY(8)='101001000000000011000000000000011000'B;
  MEMORY(9)='000001000000000111000000000000010011'B;
  MEMORY(10)='100011100000000011000000000000010111'B;
 MEMORY(11)='101001100000000011000000000000010111'B;
 MEMORY(12)='000001000000000111000000000000001111'B;
 MEMORY(13)='001010100000000000001000010100000010'B;
 MEMORY(14)='000001000000000000000000000000001010'B;
 MEMORY(15)='100011000000000011000000000000010111'B;
 MEMORY(16)='001010000000000000001101001000000011'B;
 MEMORY(17)='001011000000000000001101001000000011'B;
 MEMORY(18)='000001000000000000000000000000000110'B;
 MEMORY(19)='010000100000101010000000000000100100'B;
 MEMORY(20)='011010000000000001000000000000000001'B;
 MEMORY(21)='000001000000000000000000000000000010'B;
 MEMORY(22)= 0 ;
 MEMORY(23)='000000000000000000000000000000000001'B;
 MEMORY(24)='000000000000000000000000000000000010'B;
 MEMORY(25)='000000000000000000000000000000001010'B;
 MEMORY(26)='000000000000000000000000000000000001'B;
 MEMORY(27)='000000000000000000000000000000000010'B;
 MEMORY(28)='000000000000000000000000000000000011'B;
 MEMORY(29)='000000000000000000000000000000000100'B;
 MEMORY(30)='000000000000000000000000000000000101'B;
 MEMORY(31)='000000000000000000000000000000000110'B;
 MEMORY(32)='000000000000000000000000000000000111'B;
 MEMORY(33)='000000000000000000000000000000001000'B;
 MEMORY(34)='000000000000000000000000000000001001'B;
 MEMORY(35)='000000000000000000000000000000001010'B;
 PUT FILE(HOWARD) LIST
          ((MEMORY(I) DO I=0 TO 1023));
 END;
```

Amendment to CGTM 96

David Wyeth
October 1970

The following amendments have been made to the IOP Miniflow assembler so that an assembly source deck for this assembler will also be compatible with the ICAP II assembler, used at the Standard Computer Corporation in Santa Ana. Thus, this will facilitate testing of IOP programs in Santa Ana. A note has been added concerning control cards required to run an assembler under ICAP II and the remaining small differences between the two assemblers.

Changes Made to Miniflow Assembler

1. The mnemonic for the VFD command has been changed from 'VFD' to 'DATA'.
2. The delimiter used in the operand field of a DATA command has been changed from '/' to ':'.
3. An octal specification in a VFD command has been altered from ØNN/XXX to NN:0XXX, i.e. a number preceded by a zero indicates that the number is octal and not decimal.
4. The specification of an octal number in the operand field of an instruction has been changed from Ø/XXX to 0XXX, i.e. a number preceded by a zero indicates that the number is octal and not decimal.
5. The OCT and OCH commands have been combined into the OCT command with the option of one or two operands. The specification of one six digit octal operand (no leading zero required) causes the number to be loaded into the next available 18 bits. Two six digit octal operands separated by a ',' cause alignment to a word boundary and the insertion of the two numbers into the respective halves of the next 36 bit word.

   Examples: OCT 213675
   OCT 140001, 234567

6. NOP's are now inserted after EXIT and RETURN, if either of these instructions occurs in the first half of a word.

Notes on the ICAP II Assembler and Miniflow Loader

The ICAP II Assembler accepts either BCD or EBCDIC cards as input; the default is EBCDIC. Hence any deck used at SLAC can also be used with ICAP II, without the need for conversion to BCD.

The control card used with the Miniflow Assembler

$$LIST

must be replaced by an ICAP II control card when this assembler is being used. The ICAP II control card is punched

$IOP (beginning in column 1).

Various options which can be specified on this card are detailed in the ICAP II Language Manual (FORM 801027) pages 60-61.

The Miniflow Loader uses the first 100 (octal) words of IOP control store; thus, these locations should be used for storage only and not contain part of a miniflow program. Otherwise, the Loader will be overwritten and loading will cease. The only exception to this is that a transfer instruction to the start of the program should be assembled into location zero. The Loader branches to location zero and executes the instruction there on completion of loading.

Remaining Differences Between ICAP II and the Miniflow Assembler

1. In the Miniflow Assembler '*' may be used to signify the current value of the location counter. In ICAP II, the use of '*' in an operand field (except to indicate multiplication) is not valid. Labels must be specified to obtain a similar effect.

2. The Miniflow Assembler allows one to use a subset of the capabilities offered by ICAP II. The facilities offered by ICAP II over the Miniflow Assembler include macro expansion, arithmetic using any radix, definition of new instructions. A full description of ICAP II is given in "ICAP II Language - Programmers' Guide", Standard Computer Corporation, Form 801027.

for GTM-96

5. THE ICAP II SUPERVISOR

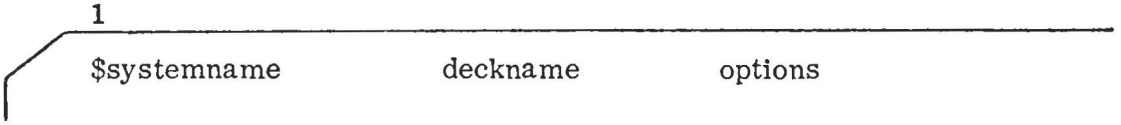
The ICAP II system may be augmented by the user to include pre-defined definitions for any number of target systems (see the ICAP II System Programmer's Guide, Form , for the method of augmenting the system). A supervisory program selects the particular set of definitions required for a given assembly. In addition, the user may specify several assembly options to the supervisor.

SUPERVISOR CONTROL CARDS

The supervisor is directed by control cards which are described in the following paragraphs.

$Systemname Card

The format of this control card is:



The $ in column 1 identifies the card as a control card. Systemname is a five character name that selects the particular pre-defined set of definitions for the assembly run. A control card punched "$ICAP" will cause the basic assembler to be selected with no pre-defined definitions. Deckname is a five character name which is retained by the system and reproduced into columns 73 thru 78 of the object deck. One or more blanks must separate "systemname" from "deckname" on the supervisor control card. A deckname is not required. One or more blanks must separate the options field from the deckname. However, if a deckname is not used, at least 11 blank columns must separate the options from the systemname. The options field is a string of key words separated by commas and terminated by a blank. The options field may be left blank. The following paragraphs describe the options. Some of the options are shown in pairs. When neither option of the pair is given on the control card, the underlined option is assumed.

{029 / 026} — 029 indicates the source deck was punched using the IBM 029 key punch character set. 026 indicates the source deck was punched using the IBM 026 key punch character set.

{DECK / NØDECK} — The DECK option indicates that an object deck is to be produced. NØDECK indicates that an object deck is not to be produced.

{PUNCH / BINTAP} — The PUNCH option indicates that the object program is to be punched on the on-line card punch. BINTAP indicates that the object is to be produced on a tape unit. These options are ignored if the NØDECK option is specified.

{LIST / BCDTAP} — The LIST option indicates that the assembly listing is to be printed with the on-line printer. BCDTAP indicates the assembly listing is to be produced on tape.

{LOAD / NØLOAD} — The LØAD option indicates that the pre-defined loader program is to precede the object deck. This option is ignored if the NØDECK option is specified or if there is no pre-defined loader for the system selected. See the ICAP II System Programmer's Guide, for the method of incorporating a loader with the defined system.

XREF — The XREF option indicates that a cross reference of label symbols used in the source program is to be printed at the end of the assembly listing.

ØPREF — The ØPREF option indicates that a cross reference of operation symbols used in the source program is to be printed at the end of the assembly listing.

SYSYM — The SYSYM option indicates that certain system symbols are to be defined. This option is used by the system programmer when preparing a new set of pre-defined definitions. See the ICAP II System Programmer's Guide for details.

ICAP II LANGUAGE
IOP MINIFLOW SUPPLEMENT

The purpose of this supplement is to describe the syntax for coding IOP (IC-7000 SPU Inner Computer) MINIFLOW. The statements which follow are included with ICAP II when the $Systemname supervisor control card is punched $IOP (beginning in column 1), indicating an IOP assembly.

WRITING MINISTEPS

The Labels Field

The labels field may be blank or may contain one or more labels. If a label is present, the location counter will be adjusted so that the ministep occupies the left half of a control storage word. A NOP ministep will be generated when necessary to fill the right half of a word.

The Operations Field

The operations field contains the symbolic operation code for the ministep. Appendix A lists the codes recognized by the assembler. Many of the operations may be suffixed with punctuation flags to indicate secondary operations. The meaning of the flags are as follows:

Flag	Meaning
.	Return, the return bit is set. When this flag is used, the following ministep will be assembled in the left half of a word.
!	Exit, the exit bit is set. When this flag is used, the following ministep will be assembled in the left half of a word.
?	Link carry and zero. The "L" bit is set for engine class instructions.

The Operands Field

The operands field contains zero, one or two operands depending on the ministep operation. See the section OPERATION CODES FOR MINIFLOW, for the operands field format.

DEFINING NEW MINISTEPS

The programmer may define MINISTEPS by using the OPVFD statement. The format of the OPVFD statement is

LABELS	OPERATIONS	OPERANDS
NAME	OPVFD	18:MASK, 18:SKEL, 3:2,1:T,32:CODE

where

NAME is the name assigned to the MINISTEP

MASK is a mask that is to be ANDed with the address of the MINISTEP

SKEL is the skeleton for the MINISTEP

T is an indicator of unconditional transfers

T=1, indicates that the MINISTEP is an unconditional transfer, an unconditional return, or an unconditional exit

CODE indicates the type of MINISTEP to be assembled as follows:

CODE	MEANING
0	Two operands are required; the first is an accumulator. The second is either an immediate operand or a scratch memory address
1	Two operands are required. The first is an accumulator. The second is an immediate operand. The assembler will insert the one's complement of the operand into the assembled MINISTEP
2	Two operands are required. The first is an accumulator number. The second is an immediate operand. The assembler will insert the two's complement of the operand into the assembled MINISTEP
3	One operand required. The operand is a scratch memory address, a channel register address, or a core memory address (IOP, CPU, or main)

4 No operands required

5 One operand required. The operand is an accumulator number

Example:

To define a ministep, SUBIM (subtract immediate), the following statement could be used:

```
SUBIM   OPVFD   18:07777,18:0400000,3:3,1:0,32:2
```

The mask 07777 will allow a 12-bit immediate operand. The assembled instruction is 400000 (add immediate). The code of 2 will cause the assembler to generate the two's complement of the immediate operand.

OPERATION CODES FOR IOP MINIFLOW

The following table gives the operation codes recognized by the assembler.

The notation used to describe operands is:

AC - the value of the operand is an accumulator number.

ADD12 - the value of the expression is the address of main storage, ALP control storage, or SPU control storage.

COUNT - the value of the operand is a shift count.

DATA - the value of the operand is immediate data.

LOCATION - the value of the expression is the location of a ministep.

REG - the value of the expression is the address of a channel or general register.

SM - the value of the expression is the address of a scratch memory cell.

NAME	OPERANDS	FLAGS	OCTAL	DESCRIPTION
ADD	AC,SM	.!?	440000	Add scratch to AC
ADDT	AC,SM	.!?	400000	Test by adding scratch to AC
ADDX	AC,SM	.!?	441000	Add scratch to AC, use target previous result indicators
ADMT	AC,DATA		000000	Test by adding immediate data to AC
AM	AC,DATA		040000	Add immediate data to AC
AND	AC,SM	.!?	540000	AND scratch to AC
ANDT	AC,SM	.!?	500000	Test by ANDing scratch with AC
ANDX	AC,SM	.!?	541000	AND scratch to AC using target previous result indicator
ANM	AC,DATA		140000	AND immediate data to AC
ANMT	AC,DATA		100000	Test by ANDing immediate data with AC
CALL	LOCATION		412000	Call subroutine
CALL*	SM		416000	Call subroutine indirectly
CLR	AC		340000	Clear the AC
COM	AC		544000	Form one's complement of AC
EXIT			472000	Exit to scheduler
HALT		.!	644000	Halt the SPU
LD	AC,SM	.!?	740000	Load AC from scratch
LD1	AC	.!	740031	Load A 1 into AC
LDM	AC,DATA		340000	Load immediate data to AC
LDX	AC,SM	.!?	741000	Load AC from scratch, use target previous result indicators
NOP	LOCATION	.!	442000	No Operation
OR	AC,SM	.!?	640000	OR scratch to AC

NAME	OPERANDS	FLAGS	OCTAL	DESCRIPTION
ORM	AC,DATA		240000	OR immediate data to AC
ORMT	AC,DATA		200000	Test by ORing immediate data with AC
ORT	AC,SM	.!?	600000	Test by ORing scratch with AC
ORX	AC,SM	.!?	641000	OR scratch to AC using target previous result indicators
RCPU	ADDR		744000	Read CPU storage
RCR	REG		774000	Read Channel Register
RET			462000	Return from subroutine
RGEN	REG	.!	635000	Read General register
RGEN*	SM	.!	635200	Read General register indirect
RIOP	ADDR		754000	Read IOP storage
RIOP*	SM	.!	764000	Read IOP storage indirect
RM	SM	.!	605000	Read Main storage indirect
RMDIR	SM	.!	605300	Read Main storage indirect to directive register
RMDN	SM	.!	605200	Read Main storage indirect to directive and N fields
RMIND	SM	.!	605100	Read Main storage indirect to indirect field
ROT	COUNT		504000	Rotate AC1 and AC2 left
ROT*			504200	Rotate AC1 and AC2 left, use present count
SCT	COUNT		514000	Shift AC1 left and count zero's
SLT	AC,COUNT		404000	Shift AC left
SLT*	AC		404200	Shift AC left, use present count
SRT	AC,COUNT		444000	Shift AC right
SRT*	AC		444200	Shift AC right, use present count

NAME	OPERANDS	FLAGS	OCTAL	DESCRIPTION
STO	AC,SM	.!	644000	Store AC into scratch
TC	LOCATION	.!	502000	Transfer on carry
TC*	SM	.!	506000	Transfer indirect on carry
TCNZ	LOCATION	.!	702000	Transfer on carry and not zero
TCNZ*	SM	.!	706000	Transfer indirect on carry and not zero
TERM		.!	644003	Set a CPU terminate request
TNC	LOCATION	.!	542000	Transfer on no carry
TNC*	SM	.!	546000	Transfer indirect on no carry
TNCUZ	LOCATION	.!	742000	Transfer on no carry or zero
TNCUZ*	SM	.!	746000	Transfer indirect on no carry or zero
TNZ	LOCATION	.!	602000	Transfer on not zero
TNZ*	SM	.!	606000	Transfer indirect on not zero
TPRI*	SM	.!		Transfer indirect on target previous result indicators
TRA	LOCATION		402000	Transfer unconditionally
TRAP		.!	644002	Set a CPU trap request
TRA*	SM		406000	Transfer indirect unconditionally
TST	SM	.!?	700000	Test scratch memory for zero
TZE	LOCATION	.!	642000	Transfer on zero
TZE*	SM	.!	646000	Transfer indirect on zero
UNHANG		.!	644001	Unhang and start the CPU
WGEN	REG	.!	634000	Write to general register
WGEN*	SM	.!	634200	Write to general register indirect
WIOP	ADDR		714000	Write to IOP storage
WIOP*	SM	.!	724000	Write to IOP storage indirect

NAME	OPERANDS	FLAGS	OCTAL	DESCRIPTION
WCPU	ADDR		704000	Write to CPU storage
WCR	REG		734000	Write to channel register
WM	SM	.!	604000	Write to Main storage
ZERO	AC	.!?	740030	Zero to AC

EXTENDED OPERATIONS

The ICAP II assembler provides operation codes to allow the programmer to specify immeidate engine, and transfer operations more conveniently.

Immediate Engine Operations

When the programmer writes one of the following immediate operations, the assembler will assemble an ADD, AND, or OR instruction with either the one's or two's complement of the immediate operand as appropriate.

NAME	OPERANDS	DESCRIPTION	EQUIVALENT INSTRUCTION
CM	AC,DATA	Compare immediate	AMT AC, -DATA
NANM	AC,DATA	Not AND immediate	ANM AC, ¬ DATA
NANMT	AC,DATA	Not AND immediate test	ANMT AC, ¬ DATA
NORM	AC,DATA	Not OR immediate	ORM AC, ¬ DATA
NORMT	AC,DATA	Not OR immediate test	ORMT AC, ¬ DATA
SM	AC,DATA	Subtract immediate	AM AC, - DATA

Transfer Operations

The extended transfer instructions are used to transfer as the result of a comparison or a subtraction. It is assumed that the programmer has conditioned the test by a sequence such as

```
LD    1,A      LOAD AC 1 FROM A
CM    1,B      COMPARE A:B
```

In the table that follows, the description will assume the comparison of A:B above.

NAME	DEFINITION	CONDITION FOR TRANSFER
TLT	Transfer less than	$A < B$
TLE	Transfer less than or equal	$A \leq B$
TEQ	Transfer equal	$A = B$
TNE	Transfer not equal	$A \neq B$
TGE	Transfer greater than or equal	$A \geq B$
TGT	Transfer greater than	$A > B$