Bcpl Runtime Package

This package is a replacement for the standard Bcpl runtime (the one built into the Alto Operating System), in which nearly all of the operations have been microprogrammed. Typical Bcpl programs run 25 to 30 percent faster than with the standard routines, depending primarily on their frequency of procedure calls and their richness in complex structure references. Use of this package also permits one to Junta to levBasic if desired, for a savings of approximately 500 words of main memory.

The microprogrammed runtime is entirely compatible with the standard one. It does not require programs to be modified or recompiled, and it works correctly during calls to the Operating System as well as to your own procedures. The simplest use of this package requires only that you load the necessary microcode into the Ram and call one initialization routine.

The package also provides a convenient framework in which to define and microprogram additional emulator opcodes.

1. Standard Use

The simplest case applies when you do not need to include any special microcode of your own. The file BcplRuntime.Dm is a dump-format file containing BcplRuntime.Br and BcplRuntimeMc.Br. These modules should be loaded with your program, along with the LoadRam procedure, available separately as LoadRam.Br.

Early during initialization, your program should execute the following:

```
     external [ LoadRam; InitBcplRuntime; RamImage ]
     if LoadRam(RamImage) eq 0 then InitBcplRuntime()
```

(LoadRam returns zero if it successfully loaded the Ram and a nonzero result otherwise, e.g., because no Ram board is installed.)

Once this has been done, the space occupied by LoadRam.Br and BcplRuntimeMc.Br may be reclaimed. BcplRuntime.Br must remain resident throughout execution of the program, but it occupies only about 150 words whereas the others consume nearly 3000.

InitBcplRuntime sets up a 'user finish procedure' (in the manner described in the O.S. manual, section 3.12), whose purpose is to restore the normal Bcpl runtime routines when the program 'finish'es for any reason. Operation of this mechanism is ordinarily invisible; however, there are two situations in which the programmer must be aware of its workings.

First, if you execute a Junta and later a CounterJunta, the CounterJunta will itself cause the standard Bcpl runtime to be restored. The later restoration performed by the BcplRuntime package will be redundant and will do no harm, but the standard (slower) Bcpl runtime will be in use once the CounterJunta has been executed.
↑L

Second, if you Junta away the standard Bcpl runtime routines themselves, you must be careful to perform initialization in the correct order. In particular, InitBcplRuntime must be called before the Junta and before any other code that sets up user finish procedures. This ensures that at 'finish' time, the cleanup procedure in the BcplRuntime package will be the last user finish procedure executed, immediately before control returns to the operating system for the final time. If this convention is not followed, a subsequent call on the Bcpl runtime would end up diving into garbage (since InitBcplRuntime saves and restores only the runtime statics, not the code).

2. Adding Your Own Microcode

In order to implement additional emulator instructions or install microcode for special devices, it is necessary to understand the workings of the package in some detail. If you don't want to do those things, you need read no further.

The source files are contained in the dump-format file BcplRuntimeSource.Dm. It includes, among other things, the following microcode source files:

BcplRuntimeMc.Mu	The top-level microcode source file, which 'includes' all the others.
EmulatorDefs.Mu	Standard label and R-register definitions useful in writing code to be run as part of the emulator task.
RamTrap.Mu	Declarations and code for dispatching all opcodes that trap into the Ram.
GetFrame.Mu	Microcode implementing the Bcpl runtime 'GetFrame' and 'Return' operations.
BcplUtil.Mu	Microcode implementing all remaining Bcpl runtime operations.

In addition to these files, you need AltoConsts23.Mu (or whatever the current version is), Mu.Run, and PackMu.Run. The latest (October 11, 1977) version of Mu is required.

To add new opcodes, you will need to edit BcplRuntimeMc.Mu and RamTrap.Mu (which should be renamed to something else first). The changes to BcplRuntimeMc.Mu are trivial: simply append 'include' statements for each of your own source files.

RamTrap.Mu contains the following predefinition:

```
    !37,40, TrapDispatch,,, GetFrame, Return, BcplUtility;
```

The labels in this predefinition correspond to the opcodes #60000, #60400, #61000, #61400, ..., #77400 (a total of 32). However, several of these cannot be used because their execution does not cause a trap into the Ram. These are #60000, #60400, #61000, #64400, #65000,
↑L

#67000, and #77400. The GetFrame, Return, and BcplUtility instructions use #61400, #62000, and #62400. All others are available for your own use simply by adding labels to the predefinition.

When one of these labels is reached, the Alto is in a clean state (no TASK or memory reference pending), the accumulators AC0 through AC3 contain the values supplied by the emulated program, and IR (the DISP bus source) contains the low-order 8 bits of the opcode, which may be used for further dispatch if desired.

The routine should finish by executing the following sequence of operations:

```
TASK;
something;
SWMODE;
:START;
```

It is essential that the TASK be executed as late as possible before the branch to START. The worst-case path in the Rom microcode beginning at START consists of 19 microinstruction cycles without a TASK. It has been determined empirically that as few as 3 microinstructions inserted between 'something' and 'SWMODE' in the above sequence causes Diablo Model 44 disks to get data-late errors. (Alas, it is not possible to say 'SWMODE, TASK' in one microinstruction because they are both F1's. In hindsight, it would have been nice if SWMODE had been implemented in such a way as to cause a TASK also.)

BcplUtil.Mu contains three convenient exit points to which opcode emulation routines may branch. The code for these exit points is:

```
Start0: PC←L;
Start1: L←PC, SWMODE;
Start2: PC←L, :START;
```

One may branch to Start0 having just executed 'L← new PC, TASK;', to Start1 having just executed 'TASK; something;', or to Start2 having just executed 'TASK; something; L← new PC, SWMODE;'.

Standard R-registers available to the routine are listed in EmulatorDefs.Mu. These are SAD, XREG, XH, MTEMP, DWAX, and MASK. All except MTEMP are used exclusively by the emulator task and may be clobbered arbitrarily (the standard Nova emulator in the Rom does not depend on them). MTEMP is usable by any task but is safe only until the next TASK.

You may need to modify EmulatorDefs.Mu if your microcode defines labels in low, fixed locations (e.g., START or the task starting addresses). Note that EmulatorDefs.Mu defines all labels except TRAP1 in a way that does not consume space in the Ram. You may need to change one or more of these (e.g., START) to ordinary predefinitions if you intend to define them in the Ram.

The microcode is assembled and turned into a .Br file by means of the commands:

```
Mu BcplRuntimeMc.Mu
PackMu BcplRuntimeMc.Mb BcplRuntimeMc.Br
```

The Bcpl runtime microcode contained in the package occupies 337 (decimal) microinstruction words.
↑L

Mu Summary

```
T ← <BUS>
  <ALU Function>*
L ← <ALU Function>
  M ←  (Sympathetic to L)
  MAR ← <ALU Function>

MD ← <BUS>

[R] ← L
  L LSH 1
  L RSH 1
  L LCY 8
[S] ← M (L)



<ALU Function> ::=
  T                    [DEST]
  <BUS>                [DEST]
  <BUS> + 1            [DEST, T]
  <BUS> - 1            [DEST, T]
  <BUS> + T            [DEST]
  <BUS> - T            [DEST]
  <BUS> + T + 1                [DEST, T]
  <BUS> - T - 1                [DEST]
  <BUS> OR T           [DEST, T]
  <BUS> AND T                  [DEST]
  <BUS>.T (AND T)      [DEST, T]
  <BUS> AND NOT T      [DEST]
  <BUS> XOR            [DEST]

<BUS> ::=
  [R]
    [S]
    [CONSTANT]
  0
  -1
  M
  MD
  MOUSE
  DISP [low order 8 bits of IR, sign extended]
```

Sequencing of Data Movements

```
<BUS> ← 1

<ALU> ← T [2]
  <BUS> [2]

MD ← <BUS>[2]
```

```
  [R] ← L [2,3] (data goes to shifter on cycle 2)
    L LSH 1 [2,3]
    L RSH 1 [2,3]
    L LCY 8 [2,3]

  T ← <BUS>[3]
    <ALU Function>*[3]

  L ← <ALU Function>[3]
    M ←  (Sympathetic to L)
    MAR ← <ALU Function>[3]

  [S] ← M (L) [3]
```

Compatible Simultaneous Data Movements

```
 <BUS>←
   T←
   L←
   MAR←
 T←
   <BUS>← or [R]← or[S]←
   L←
   MAR←
   MD←

 MAR←
   <BUS>← or [R]← or [S]←
   L←
   T←

 MD←
   T←
   L←

 L←
   MAR←
   <BUS>←
   T←

 [R]←L
   T,L,MAR←<ALU Function (BUS ← 0,T)>

 [R]←L <SHIFT OPERATION>
   T,L←<ALU Function (BUS ← 0,T)>

 [S]←L
   T,L,MAR←<ALU Function (BUS ← 0,T)>
```

```
;       A L T O C O N S T S 2 3 . M U

; Symbol and constant definitons for the standard Alto microcode.
; These definitions are for:
;       AltoCode23, AltoCode24, AltoIICode2, and AltoIICode3
; By convention, people writing microcode should 'include' this file
;   in front of their microcode using the following MU construct:
;       #AltoConsts23.mu;
; This entire file is full of magic.  If you modify it in any way
;   you run the risk of being incompatible with the Alto world,
;   not to mention having your Alto stop working.
;
; Revision History:
; September 20, 1977  8:33 PM by Boggs
;       Created from old AltoConsts23.mu
; September 23, 1977  12:17 PM by Taft
; October 11, 1977  2:07 PM by Boggs
;       Added XMAR definition
;
```

```
;Symbol definitons

;Bus Sources
;BS 0 ← RRegister
;BS 1 zeros the bus during RRegister←, BUT NOT SRegister←
;BS 2 is undefined and therefore makes the bus all ones
;BS 3 and 4 are task specific.  For the 'Ram related' tasks they are:
;       BS 3: ← SRegister
;       BS 4: SRegister←
;BS 5 is main memory (see definiton for MD, below)
$MOUSE          $L000000,014006,000100; BS = 6
$DISP           $L000000,014007,000120; BS = 7

;Standard F1s
$XMAR           $L072000,000000,144000; F1 = 1 and F2 = 6 (Extended MAR)
$MAR            $L020001,000000,144000; F1 = 1
$TASK           $L016002,000000,000000; F1 = 2
$BLOCK          $L016003,000000,000000; F1 = 3
$LLSH1          $L000000,022004,000200; F1 = 4
$LRSH1          $L000000,022005,000200; F1 = 5
$LLCY8          $L000000,022006,000200; F1 = 6

;Standard F2s
$BUS=0          $L024001,000000,000000; F2 = 1
$SH<0           $L024002,000000,000000; F2 = 2
$SH=0           $L024003,000000,000000; F2 = 3
$BUS            $L024004,000000,000000; F2 = 4
$ALUCY          $L024005,000000,000000; F2 = 5
$MD             $L026006,014005,124100; F2 = 6, BS = 5

;Emulator specific functions
$BUSODD         $L024010,000000,000000; F2 = 10
$LMRSH1         $L000000,062005,000200;     F2 = 11 Magic Right Shift
$LMLSH1         $L000000,062004,000200;     F2 = 11 Magic Left Shift
$DNS            $L030012,000000,060000; F2 = 12 Do Nova Shift
$ACDEST         $L030013,032013,060100; F2 = 13 Nova Destination AC
$IR             $L026014,000000,124000; F2 = 14 Instruction Register
$IDISP          $L024015,000000,000000; F2 = 15 IR Dispatch
$ACSOURCE       $L000000,032016,000100; F2 = 16 Nova Source AC

;Emulator specific functions decoded by the RAM board
$SWMODE         $L016010,000000,000000;     F1 = 10 Switch Mode
$WRTRAM         $L016011,000000,000000;     F1 = 11 Write Ram
$RDRAM          $L016012,000000,000000;     F1 = 12 Read Ram
$RMR            $L020013,000000,124000;     F1 = 13 Reset Mode Register
;F1 = 14 and 15 are used by the magic shifts

;Emulator specific functions decoded by the ETHERNET board
$RSNF           $L000000,070016,000100;     F1 = 16 Read Serial (Host) Number
$STARTF         $L016017,000000,000000;     F1 = 17 Start I/O

$M              $R40;                   The M Register
$L              $L040001,036001,144200; The L Register
$T              $L052001,054001,124040; ALUF = 1, The T Register

;ALU Functions.  * => loads T from ALU output
$ORT            $L000000,050002,000002; ALUF = 2 *
$ANDT           $L000000,050003,000002; ALUF = 3
$XORT           $L000000,050004,000002; ALUF = 4
```

```
$+1             $L000000,050005,000002; ALUF = 5 *
$-1             $L000000,050006,000002; ALUF = 6 *
$+T             $L000000,050007,000002; ALUF = 7
$-T             $L000000,050010,000002; ALUF = 10
$-T-1           $L000000,050011,000002; ALUF = 11
$+INCT                  $L000000,050012,000002; ALUF = 12 * synonym for +T+1
$+T+1           $L000000,050012,000002; ALUF = 12 *
$+SKIP                  $L000000,050013,000002; ALUF = 13
$.T             $L000000,050014,000002; ALUF = 14 *
$AND NOT T $L000000,050015,000002; ALUF = 15
;$ZEROALU       $L000000,050016,000040;     ALUF = 16
;ALUF 17 is unassigned

;Handy fakes
$SINK           $L044000,000000,124000;     DF3 = 0  Bus source without dest
$NOP            $L042000,000000,000000;     NDF3 = 0 every computer needs one

; Definitions for the Nova debugger and DEBAL
$HALT           $L042001,000000,000000;
$BREAK                  $L042003,000000,000000;
$WENB                   $L042005,000000,000000;
$READY?                 $L042006,000000,000000;
$NOVA                   $L044002,046003,124100;
$END            $L034000,000000,000000;
```

```
;Constant definitions

$0              $L000000,012000,000100;       Constant 0 is SUPER SPECIAL

$ALLONES4       $M4:177777;     Constant normally ANDed with KSTAT
$ALLONES5       $M5:177777;     Constant normally ANDed with MD
$M17            $M6:000017;     Constant normally ANDed with MOUSE
$ALLONES7       $M7:177777;     Constant normally ANDed with DISP
$M177770        $M7:177770;     Mask for DISP
$M7             $M7:000007;     Mask for DISP
$X17            $M7:000017;     Mask for DISP

$ONE            $1;             The constant 1
$2              $2;
$-2             $177776;        - Disk header word count
$3              $3;
$4              $4;
$5              $5;
$6              $6;
$7              $7;
$10             $10;
$-10            $177770;        - Disk label word count
$17             $17;
$20             $20;
$37             $37;
$ALLONES        $177777;        The REAL -1 (not a mask)
$40             $40;
$77             $77;
$100            $100;
$177            $177;
$200            $200;
$377            $377;
$177400                 $177400;
$-400           $177400;        - DISK DATA WORD COUNT
$2000           $2000;
$PAGE1                  $400;
$DASTART        $420;           MAIN MEMORY DISPLAY HEADER ADDRESS
$KBLKADR        $521;           MAIN MEMORY DISK BLOCK ADDRESS
$MOUSELOC       $424;           MAIN MEMORY MOUSE BLOCK ADDRESS
$CURLOC                 $426;           MAIN MEMORY CURSOR BLOCK ADDRESS
$CLOCKLOC       $430;
$CON100                 $100;
$CADM                   $7772;          CYLINDER AND DISK MASK
$SECTMSK        $170000;        SECTOR MASK
$SECT2CM        $40000;         CAUSES ILLEGAL SECTORS TO CARRY OUT
$-4             $177774;        CURRENTLY UNUSED
$177766                 $177766;        CURRENTLY UNUSED
$177753                 $177753;        CURRENTLY UNUSED
$TOTUWC                 $44000;         NO DATA TRANSFER, USE WRITE CLOCK
$TOWTT                  $66000;         NO DATA TRANSFER, DISABLE WORD TASK
$STUWC                  $4000;          TRANSFER DATA USING WRITING CLOCK
$STRCWFS        $10000;         TRANSFER DATA USING NORMAL CLOCK, WAIT FOR SYNC
$177000                 $177000;
$77777          $77777;
$77740          $77740;
$LOW14                  $177774;
$77400          $77400;
$-67D           $177675;
$7400           $7400;
```

```
$7417           $7417;
$170360                 $170360;
$60110          $60110;
$30000          $30000;
$70531          $70531;
$20411          $20411;
$65074          $65074;
$41023          $41023;
$122645                 $122645;
$177034                 $177034;
$37400          $37400;
$BIAS           $177700;        CURSOR Y BIAS
$WWLOC                  $452;           WAKEUP WAITING IN PAGE 1
$PCLOC                  $500;           PC VECTOR IN PAGE 1
$100000                 $100000;
$177740                 $177740;
$COMERR1        $277;           COMMAND ERROR MASK
$-7             $177771;        CURRENTLY UNUSED
$177760                 $177760;
$-3             $177775;
$4560           $4560;
$56440          $56440;
$34104          $34104;
$64024          $64024;
$176000                 $176000;
$177040                 $177040;
$177042                 $177042;
$203            $203;
$360            $360;
$177600                 $177600;
$174000                 $174000;
$160000                 $160000;
$140000                 $140000;
$777            $777;
$1777           $1777;
$3777           $3777;
$7777           $7777;
$17777          $17777;
$37777          $37777;
$1000           $1000;
$20000          $20000;
$40000          $40000;
$-15D           $177761;
$TRAPDISP       $526;
$TRAPPC                 $527;
$TRAPCON        $470;
$JSRC           $6000;          JSR@ 0
$MASKTAB        $460;           Mask Table Starting address for convert
$SH3CONST       $14023;         DESTINATION = 3, SKIP IF NONZERO CARRY,
;                               BASE CARRY = 0

$600            $600;           Ethernet addresses
$601            $601;
$602            $602;
$603            $603;
$604            $604;
$605            $605;
$606            $606;
$607            $607;
$610            $610;
```

```
$612            $612;

$ITQUAN                 $422;
$ITIBIT                 $423;
$402            $402;           where label block is stored on disk boot
$M177760        $M7:177760;     MASK FOR DISP.  FOR I/O INSTRUCTIONS
$JSRCX                  $4000;          JSR 0
$KBLKADR2       $523;
$KBLKADR3       $524;

$MFRRDL                 $177757;        DISK HEADER READ DELAY IS 21 WORDS
$MFR0BL                 $177744;        DISK HEADER PREAMBLE IS 34 WORDS
$MIRRDL                 $177774;        DISK INTERRECORD READ DELAY IS 4 WORDS
$MIR0BL                 $177775;        DISK INTERRECORD PREAMBLE IS 3 WORDS
$MRPAL                  $177775;        DISK READ POSTAMBLE LENGTH IS 3 WORDS
$MWPAL                  $177773;        DISK WRITE POSTAMBLE LENGTH IS 5 WORDS
$BDAD           $12;            ON BOOT, DISK ADDRESS GOES IN LOC 12

$REFMSK                 $77740;         MRT Refresh mask
$X37            $M7:37;                 NOPAR MASK
$M177740        $M7:177740;     DITTO
$EIALOC                 $177701;        LOCATION OF EIA INPUT HARDWARE

$7000           $7000;          mapbase
$176            $176;           mapmask
$177576                 $177576;        mapmask3
$30             $30;            reprobinc
$15             $15;            wrt-1
$1770           $1770;          ciad
$101771                 $101771;        cilow
$175777                 $175777;        for resetting fbn
$11             $11;            just to have small integers
$13             $13;
$14             $14;
$16             $16;            for 2CODE
$60             $60;            low R to high R bus source
$776            $776;
$177577                 $177577;        -129
$100777                 $100777;
$177677                 $177677;
$177714                 $177714;        (-2fvar+14)

$2527           $2527;
$101            $101;
$630            $630;
$631            $631;
$642            $642;

$lgm1           $M7:1;
$lgm3           $M7:3;
$lgm10          $M7:10;
$lgm14          $M7:14;
$lgm20          $M7:20;
$lgm40          $M7:40;
$lgm100                 $M7:100;
$lgm200                 $M7:200;

$disp.300       $M7:300;
$-616           $177162;
$-650           $177130;
```

```
$22           $22;
$24           $24;
$-20          $177760;
$335          $335;           endcode for getframe
$1377         $1377;          smallnzero
$401          $401;
$2001         $2001;
$21           $21;            just to have them
$23           $23;
$25           $25;
$26           $26;
$27           $27;
$31           $31;
$1675         $1675;
$736          $736;
$-660         $177120;
$300          $300;
$disp.377     $M7:377;
$6001         $6001;          f.e. flg, quick flg, use count
$disp.3       $M7:3;

; Constants for subroutine returns using IR.
; See 9.2.1 of the hardware manual for details.

$sr1          $60110;
$sr0          $70531;
$sr2          $61000;
$sr3          $61400;
$sr4          $62000;
$sr5          $62400;
$sr6          $67000;         value of 16b mapped to 6 by disp prom
$sr7          $63400;
$sr10         $64024;
$sr11         $64400;
$sr12         $65074;
; Are you wondering why sr13 is missing?  So is everyone else.
$sr14         $66000;
$sr15         $66400;
$sr16         $63000;         value of 6 mapped to 16b by disp prom
$sr17         $77400;
$sr20         $65400;
$sr21         $65401;
$sr22         $65402;
$sr23         $65403;
$sr24         $65404;
$sr25         $65405;
$sr26         $65406;
$sr27         $65407;
$sr30         $65410;
$sr31         $65411;
$sr32         $65412;
$sr33         $65413;
$sr34         $65414;
$sr35         $65415;
$sr36         $65416;
$sr37         $65417;

$-13D         $177763;

$ERRADDR      $177024;        AltoII MEAR (Memory Error Address Reg)
```

```
$ERRSTAT       $177025;         AltoII MESR (Memory Error Status Reg)
$ERRCTRL       $177026;         AltoII MECR (Memory Error Control Reg)
$REFZERO       $7774;

$2377          $2377;           Added for changed Ethernet microcode
$2777          $2777;
$3377          $3377;
$477           $477;            Added for BitBlt
$576           $576;            Added for Ethernet boot
$177175                 $177175;

;Requests for the following new constants have been made:
;NOTE THAT THESE ARE NOT YET DEFINED

;$lgm2         $M7:2;
;$lgm4         $M7:4;
;$32           $32;
;$33           $33;
;$34           $34;
;$35           $35;
;$36           $36;
```

```
; RamTrap.mu

;       Last modified October 11, 1977  11:26 AM

; Trap handler and dispatcher for instructions that trap into
; the RAM.  In the following predefinition, the tags correspond
; to opcodes 60000, 60400, 61000, 61400, ... 77400.
; Note that opcodes 60000, 60400, 61000, 64400, 65000, 67000, and 77400
; cannot be used since control never gets to the RAM for these).

;                       61400     62000    62400
!37,40, TrapDispatch,,, GetFrame, Return, BcplUtility;


; Control comes here with the instruction LCY 8 in XREG
TRAP1:T←37;
        L←XREG AND T;
TrapDispatch:
        SINK←LREG, BUS, TASK;
        :TrapDispatch;
```

```
; EmulatorDefs.mu -- Alto definitions for emulator-level microprogramming

;       Last modified October 12, 1977  2:57 PM


; Standard microinstruction addresses in the Rom
; (see Alto Hardware Manual, section 9.1).
; These declarations do not cause space to be allocated in the Ram
; (except TRAP1, which we presumably want to define in the Ram)

$START          $L4020, 0, 0;
$RAMCYCX        $L4022, 0, 0;
$BLT            $L4105, 0, 0;
$BLKS           $L4106, 0, 0;
$MUL            $L4120, 0, 0;
$DIV            $L4121, 0, 0;
$BITBLT         $L4124, 0, 0;
$L0             $L4160, 0, 0;

!37,1, TRAP1;


; Standard R-registers usable by the emulator task

$AC3            $R0;    Accumulators
$AC2            $R1;
$AC1            $R2;
$AC0            $R3;
$NWW            $R4;    New wakeups waiting (communication between tasks)
$SAD            $R5;    Temporary private to emulator
$PC             $R6;    Program Counter for emulated Nova
$XREG           $R7;    Temporary private to emulator.
;                       Contains instruction LCY 8 upon dispatch to TRAP1.
$XH             $R10;   Temporary private to emulator
$MTEMP          $R25;   Temporary usable by any task
$DWAX           $R35;   Temporary private to emulator
$MASK           $R36;   Temporary private to emulator

$LREG           $R40;   Another name for the M-register
```

```
; BcplRuntimeMc.mu -- top-level microprogram for Bcpl runtime code

; Last modified October 11, 1977  11:25 AM

#AltoConsts23.mu;
#EmulatorDefs.mu;
#RamTrap.mu;
#GetFrame.mu;
#BcplUtil.mu;
```

```
; BcplUtil.Mu  --  bcpl runtime utilities (except GetFrame and Return)

;       Last modified October 16, 1977  6:38 PM


; All Bcpl runtime utilities in this module are invoked by an opcode
; of the form XXnnn, where XX is the opcode for the main dispatch in RamTrap
; and nnn is the DISP field used for sub-dispatching here.

!77,100, Lq0.0, Lq0.1, Lq0.2, Lq0.3, Lq0.4, Lq0.5, Lq0.6, Lq0.7,
                , Lq1.1, Lq1.2, Lq1.3, Lq1.4, Lq1.5, Lq1.6, Lq1.7,
        Snq0, Sq0.1, Sq0.2, Sq0.3, Sq0.4, Sq0.5, Sq0.6, Sq0.7,
        Snq1, Sq1.1, Sq1.2, Sq1.3, Sq1.4, Sq1.5, Sq1.6, Sq1.7,
        LongJump, Branch, Lookup, Rsh, Lsh, Ior, Xor, Eqv,
        Mult, DivRem, MulPlus, Ly01, Ly10, Sy01, Sy10;


; RamTrap dispatches here for the Bcpl utility opcode

BcplUtility:
        SINK←DISP, BUS, TASK;           Branch on sub-code
        :Lq0.0;


; LongJump
; Jumps to AC3 + @AC3
; Calling sequence is:
;       jsr @355
;        target-. (i.e., a self-relative pointer)

LongJump:
        MAR←T←AC3;
LongJ1:NOP;
        L←MD+T, TASK;

; Some useful exit sequences
Start0: PC←L;                   Branch here having done L← new PC, TASK;
Start1: L←PC, SWMODE;                   Here after TASK; something;
Start2: PC←L, :START;                   Here after TASK; something; L← new PC, SWMODE;
```

```
; Branch
; Calling sequence is:
;       lda 0 switchon value
;       jsr @350
;        value of last case
;        number of cases
;        lastTarget-.
;        ...
;        firstTarget-.
;       return here if out of range, AC0 unchanged

!1,2, Bran0, Bran1;
!1,2, Bran2, Bran3;

Branch: MAR←T←AC3;          Fetch value of last case
        L←2+T;
        AC3←L;              AC3← address of first branch table entry
        T←AC0;              Value we are branching on
        L←MD-T;                    L← lastCase-value, carry← lastCase ge value
        MAR←T←AC3-1, ALUCY;  Fetch number of cases
        T←LREG, L←LREG+T, :Bran0; [Bran0, Bran1] T← lastCase-value,
;                           L← AC3+(lastCase-value)-1

; Value greater than last case, take out of range exit.
Bran0:  L←T←MD, :Bran1a;    Finish fetch of numCases, turn off ALUCY

; Value le last case, test number of cases
Bran1:  SAD←L;                     Save address-1 of branch table entry
        L←MD-T-1, T←MD;            L← numCases-(lastCase-value)-1, T← numCases
Bran1a: L←AC3+T, ALUCY, TASK;      Carry if numCases gr (lastCase-value)
        AC3←L, :Bran2;      [Bran2, Bran3] Adr of inst after branch table

; Value in range, execute branch.
; SAD/ address-1 of branch table entry
Bran3:  MAR←T←SAD+1, :LongJ1;      Just like LongJump

; Value less than first case, take out of range exit.
Bran2:  L←AC3, SWMODE, :Start2;
```

```
; Lookup
; Calling sequence is:
;       lda 0 switchon value
;       jsr @351
;        number of cases
;        case value 1
;        target1-.
;        ...
;        case value n
;        targetn-.
;       return here if out of range

!1,2, Look0, Look1;
!1,2, Look2, Look3;

Lookup:MAR←T←AC3;           Fetch number of cases
        NOP;
        L←MD+T, T←MD;               L← AC3+numCases, T← numCases
        L←LREG+T+1, TASK;  L← AC3+(2*numCases)+1
        AC1←L;              Save for end test

Look0:  MAR←T←AC3+1;                Increment pointer, fetch next case value
        L←AC1-T;            Test for end
        T←AC0, L←T, SH=0;   T← switchon value
        AC3←L, :Look2;              [Look2, Look3]
Look2:  L←MD-T;                     Compare switchon value with case
        L←AC3+1, SH=0, TASK;        Increment pointer again
        AC3←L, :Look0;              [Look0, Look1]

; Found matching case value.  AC3/ address of dispatch for case.
Look1:  MAR←T←AC3, :LongJ1;         Just like LongJump

; Lookup failed.  AC3/ adr of inst after lookup table
Look3:  L←AC3, TASK, :Start0;
```

```
; Right shift
; Computes ac0 ← ac0 rshift ac1
; Called by jsr @347
; Note that shift count may be either positive or negative

!1,2, RshPos, RshNeg;
!1,2, RshG16, RshL16;
!1,2, RshG8, RshL8;
!1,1, RshN1;
!1,1, LtoAC0;

Rsh:    L←T←AC1;                Shift count negative?
        L←17-T, SH<0;           16 or greater?
        L←10 AND T, ALUCY, :RshPos; [RshPos, RshNeg] 8 or greater?
RshPos:L←7 AND T, SH=0, :RshG16; [RshG16, RshL16] Compute count mod 8
RshL16:         T←177400, :RshG8;     [RshG8, RshL8]

; Shift count in range 8 to 15.  Start by right-shifting 8
RshG8: T←AC0.T;
        SINK←LREG, L←T, BUS, TASK;  Branch on shift count mod 8
        AC0←L LCY 8, :Lq0.0;

; Shift count less than 8.  Branch on shift count
RshL8: SINK←AC1, BUS, TASK;
        :Lq0.0;

; This shift table is also used in the Lq0.n series of instructions
Lq0.7:  L←AC0;
        AC0←L RSH 1;
Lq0.6:  L←AC0;
        AC0←L RSH 1;
Lq0.5:  L←AC0;
        AC0←L RSH 1;
Lq0.4:  L←AC0;
        AC0←L RSH 1;
Lq0.3:  L←AC0;
        AC0←L RSH 1;
Lq0.2:  L←AC0;
        AC0←L RSH 1;
Lq0.1:  L←AC0, TASK;
        AC0←L RSH 1, :Bran2;Do PC←AC3 and go to START

; Shift count 0, do nothing
Lq0.0:  L←AC3, SWMODE, :Start2;    Do PC←L and go to START

; Shift count 16 or greater, return zero
RshG16:         L←0, TASK, :LtoAC0;  [LtoAC0, LtoAC0]
LtoAC0:         AC0←L, :Bran2;        Do PC←AC3 and go to START

; Shift count negative.  Convert to Left Shift
RshNeg:         L←0-T, TASK;          [RshN1, RshN1] Negate shift count
RshN1: AC1←L, :Lsh;
```

```
; Right shift constant amount
; Computes ac0 ← ac0 rshift n (n in range 1 to 7)
; Calling sequence is:
;       lda 0 value
;       jsr 314 - 2*n
; (dispatches into Lq0.n table, above)


; Right shift constant amount
; Computes ac1 ← ac1 rshift n (n in range 1 to 7)
; Calling sequence is:
;       lda 1 value
;       jsr 315 - 2*n

Lq1.7:  L←AC1;
        AC1←L RSH 1;
Lq1.6:  L←AC1;
        AC1←L RSH 1;
Lq1.5:  L←AC1;
        AC1←L RSH 1;
Lq1.4:  L←AC1;
        AC1←L RSH 1;
Lq1.3:  L←AC1;
        AC1←L RSH 1;
Lq1.2:  L←AC1;
        AC1←L RSH 1;
Lq1.1:  L←AC1, TASK;
        AC1←L RSH 1, :Bran2;Do PC←AC3 and go to START
```

```
; Left shift
; Computes ac0 ← ac0 lshift ac1
; called by jsr @346
; Note that shift count may be either positive or negative

!1,2, LshPos, LshNeg;
!1,2, LshG16, LshL16;
!1,2, LshG8, LshL8;
!7,10, Lsh0, Lsh1, Lsh2, Lsh3, Lsh4, Lsh5, Lsh6, Lsh7;
!1,1, LshN1;

Lsh:    L←T←AC1;                Shift count negative?
        L←17-T, SH<0;           16 or greater?
        L←10 AND T, ALUCY, :LshPos; [LshPos, LshNeg] 8 or greater?
LshPos: L←7 AND T, SH=0, :LshG16; [LshG16, LshL16] Compute count mod 8
LshL16: T←377, :LshG8;          [LshG8, LshL8]

; Shift count in range 8 to 15.  Start by left-shifting 8
LshG8:  T←AC0.T;
        SINK←LREG, L←T, BUS, TASK;  Branch on shift count mod 8
        AC0←L LCY 8, :Lsh0;

; Shift count less than 8.  Branch on shift count
LshL8:  SINK←AC1, BUS, TASK;
        :Lsh0;

Lsh7:   L←AC0;
        AC0←L LSH 1;
Lsh6:   L←AC0;
        AC0←L LSH 1;
Lsh5:   L←AC0;
        AC0←L LSH 1;
Lsh4:   L←AC0;
        AC0←L LSH 1;
Lsh3:   L←AC0;
        AC0←L LSH 1;
Lsh2:   L←AC0;
        AC0←L LSH 1;
Lsh1:   L←AC0, TASK;
        AC0←L LSH 1, :Bran2;Do PC←AC3 and go to START

; Shift count 0, do nothing
Lsh0:   L←AC0, TASK, :LtoAC0;

; Shift count 16 or greater, return zero
LshG16:         L←0, TASK, :LtoAC0;  [LtoAC0, LtoAC0]

; Shift count negative.  Convert to Right Shift
LshNeg:         L←0-T, TASK;          [LshN1, LshN1] Negate shift count
LshN1:  AC1←L, :Rsh;
```

```
; Ior
; Computes ac0 ← ac0 % ac1
; Called by jsr @340

Ior:    T←AC1;
        L←AC0 OR T, TASK, :LtoAC0;


; Xor
; Computes ac0 ← ac0 xor ac1
; Called by jsr @341

Xor:    T←AC1;
Xor1:   L←AC0 XOR T, TASK, :LtoAC0;


; Eqv
; Computes ac0 ← ac0 eqv ac1
; Called by jsr @342

Eqv:    T←AC1;
        L←ALLONES XOR T; ac0 eqv ac1 = ac0 xor (not ac1)
        T←LREG, :Xor1;


; MulPlus
; Computes ac0 ← ac3 ← (ac1*@ac3)+ac0
; Calling sequence is:
;       lda 0 addend
;       lda 1 multiplicand
;       jsr @357
;        multiplier
;       return here with result in ac0 and ac3

!1,2, MPNoAd, MPAdd;
!1,2, MPLoop, MPDone;

MulPlus:
        MAR←AC3;            Start fetch of multiplier
        L←AC3+1;            Compute return pc
        PC←L;
        L←MD, BUSODD, :MPLp1;    Test low bit of multiplier

; MulPlus loop.  During each iteration, the multiplier is right-shifted 1
; and the multiplicand is left-shifted 1.  The loop terminates when the
; multiplier becomes zero.  This is good because in the standard use of
; MulPlus the multiplier is typically a small integer.
MPLoop:     L←AC3, BUSODD;          Test low bit of multiplier
MPLp1:      AC3←L RSH 1, :MPNoAd;   [MPNoAd, MPAdd] Shift it out

; Multiplier bit was 0, don't add but just shift multiplicand
MPNoAd:     L←AC1, SH=0, TASK, :MPShft; Test for no more bits in multiplier

; Multiplier bit was 1, add multiplicand to product
MPAdd:      T←AC1;                  Multiplicand
        L←AC0+T;            Add to partial product
        AC0←L, L←T, TASK;   L← multiplicand
MPShft:     AC1←L LSH 1, :MPLoop;   [MPLoop, MPDone] Shift multiplicand left
```

```
; Here when done
MPDone:     L←AC0, SWMODE;           Copy result to ac3
        AC3←L, :START;
```

```
; Mult
; Computes (ac0,ac1) ← ac0*ac1
; Called by jsr @343

!1,2, DoMul, NoMul;
!1,2, MNoAdd, MAdd;
!1,2, NoSpil, Spill;
!1,2, MultLp, MultDn;

Mult:   L←AC0-1, BUS=0;            Get multiplicand-1, test for zero
        SAD←L, L←0, :DoMul; [DoMul, NoMul] Save it away
DoMul:AC0←L, TASK;          Init partial product to 0
        IR←ONE;                    Init loop count; done when it reaches 20

; Multiply loop
MultLp:         L←AC1, BUSODD;             Test low bit of multiplier
        T←AC0, :MNoAdd;            [MNoAdd, MAdd] Get partial product

; Multiplier bit was 1, add multiplicand to product
MAdd:  L←T←SAD+T+1;                Add multiplicand to partial product
        L←AC1, ALUCY;              Low part of partial product

; Multiplier bit was 0, just shift multiplicand and partial product
MNoAdd:         AC1←L MRSH 1, L←T, T←0, :NoSpil; [NoSpil, Spill]
Spill:  T←ONE;                     Carry into high partial product
NoSpil: AC0←L MRSH 1;
        L←DISP+1, L←X17+1, BUS=0, TASK; Check and update loop count
        IR←LREG, :MultLp;    [MultLp, MultDn] Branch if it was 20

; Here when done
MultDn:         L←AC3, SWMODE, :Start2;

; Here when multiplicand is zero, just return zero
NoMul:AC1←L, :Bran2;
```

```
; DivRem
; Computes ac1 ← ac0/ac1 and ac0 ← ac0 rem ac1 (signed)
; Called by jsr@344 or jsr@345

!1,2, DvsPos, DvsNeg;
!1,2, DndPos, DndNeg;
!1,2, NoSub, DoSub;
!1,2, DivLp, DivDn;
!1,2, RemPos, RemNeg;
!1,2, QuoPos, QuoNeg;

DivRem:         L←T←AC1;                Fetch divisor
        SAD←L, SH<0;            Save it, test sign
        XREG←L, L←0-T, :DvsPos;     [DvsPos, DvsNeg] Save original divisor
DvsNeg:         SAD←L;                          Negative, negate divisor
DvsPos: L←T←AC0;                Fetch dividend
        PC←L, L←0-T, SH<0;  Save it, test sign
        :DndPos;                [DndPos, DndNeg] Init loop count
DndNeg:         T←LREG;                         Negative, negate dividend
DndPos:         L←20;                   Init loop count
        XH←L, L←0, :DivLp0; Init high dividend

; Divide loop
DivLp:  L←AC0;                  Current high dividend
        T←AC1;                  Current low dividend and quotient
DivLp0:AC0←L MLSH 1, L←T;Shift another bit into high dividend
        AC1←L LSH 1;            Shift a zero into quotient
        T←SAD;                          Divisor
        L←AC0-T, T←AC0;                 Try to subtract divisor from high dividend
        AC0←L, ALUCY;                   Store dividend assuming subtract ok
        L←XH-1, :NoSub;         [NoSub, DoSub] Decrement and test loop count

; Subtract ok, put a 1 in the quotient
DoSub: XH←L;                    Update loop count
        L←AC1+1, SH=0, TASK;        Change quotient bit to 1
        AC1←L, :DivLp;                  [DivLp, DivDn] Branch if done

; Subtract not ok, restore old dividend and leave quotient bit 0
NoSub: XH←L, L←T, SH=0, TASK;       Update loop count
        AC0←L, :DivLp;                  [DivLp, DivDn] Restore AC0, branch if done

; Here when done.  Fix up signs and exit
DivDn: L←PC;                    Get original dividend
        T←AC0, SH<0;            Test sign
        L←0-T, T←0, :RemPos; [RemPos, RemNeg]
RemNeg:         AC0←L, T←0-1;           Was negative, negate remainder
RemPos:         L←XREG XOR T;                   Get divisor sign, xor with dividend
        T←AC1, SH<0;            Test sign
        L←0-T, TASK, :QuoPos;
QuoNeg:         AC1←L, :Bran2;          Negate quotient
QuoPos:         :Bran2;                 Set PC←AC3 and go to START
```

```
; Sq0
; Left shifts data a constant amount, then stores in partial-word field
; in same manner as Snq0.
; Executes @ac1 ← (@ac1 & not @ac3) + ((ac0 lshift n) & @ac3)
; Calling sequence is:
;       lda 0 value (right-justified)
;       lda 1 address of word being stored into
;       jsr 333 - 2*n (n is number of left shifts desired, in range 0-7)
;        mask word (ones in field being stored into, zeroes elsewhere)
;       returns here

Sq0.7:  L←AC0;
        AC0←L LSH 1;
Sq0.6:  L←AC0;
        AC0←L LSH 1;
Sq0.5:  L←AC0;
        AC0←L LSH 1;
Sq0.4:  L←AC0;
        AC0←L LSH 1;
Sq0.3:  L←AC0;
        AC0←L LSH 1;
Sq0.2:  L←AC0;
        AC0←L LSH 1;
Sq0.1:  L←AC0, TASK;
        AC0←L LSH 1, :Snq0;


; Snq0
; Stores partial-word field into a structure.
; Executes @ac1 ← (@ac1 & not @ac3) + (ac0 & @ac3)
; Calling sequence is:
;       lda 0 value (must be bit-aligned with field being stored into)
;       lda 1 address of word being stored into
;       jsr @360
;        mask word (ones in field being stored into, zeroes elsewhere)
;       returns here

Snq0:   MAR←AC3;                Fetch mask
        L←AC1;                  Address of word being stored into
Snq0a:  T←MD;
        MAR←LREG;               Fetch word being stored into
        AC1←L;                  Save address (in case came from Snq1)
        L←MD AND NOT T;                 Zero bits to be changed
        MAR←AC1;                Start to store back updated word
        T←AC0.T;                Mask out extraneous bits in new value
        L←LREG+T, TASK;                 Merge new bits into old word
        MD←LREG;                Store back in memory
        L←AC3+1, SWMODE, :Start2; PC←AC3+1 and go to START
```

```
; Sq1
; Left shifts data a constant amount, then stores in partial-word field
; in same manner as Snq1.
; Executes @ac0 ← (@ac0 & not @ac3) + ((ac1 lshift n) & @ac3)
; Calling sequence is:
;       lda 1 value (right-justified)
;       lda 0 address of word being stored into
;       jsr 334 - 2*n (n is number of left shifts desired, in range 0-7)
;        mask word (ones in field being stored into, zeroes elsewhere)
;       returns here

Sq1.7:  L←AC1;
        AC1←L LSH 1;
Sq1.6:  L←AC1;
        AC1←L LSH 1;
Sq1.5:  L←AC1;
        AC1←L LSH 1;
Sq1.4:  L←AC1;
        AC1←L LSH 1;
Sq1.3:  L←AC1;
        AC1←L LSH 1;
Sq1.2:  L←AC1;
        AC1←L LSH 1;
Sq1.1:  L←AC1, TASK;
        AC1←L LSH 1, :Snq1;


; Snq1
; Stores partial-word field into a structure.
; Executes @ac0 ← (@ac0 & not @ac3) + ac1 & @ac3
; Calling sequence is:
;       lda 1 value (must be bit-aligned with field being stored into)
;       lda 0 address of word being stored into
;       jsr @360
;        mask word (ones in field being stored into, zeroes elsewhere)
;       returns here

Snq1:   MAR←AC3;             Fetch mask
        L←AC1;               Get value
        T←AC0;               Get address
        AC0←L, L←T, :Snq0a;  Swap them and join common code
```

```
; Load byte from array
; Loads the ac1'th byte from the array pointed to by ac0
; and returns it right-justified in ac0.
; Called by jsr @362
; Note:  ac1 may be negative.

!1,2, Ly01P, Ly01N;
!1,2, Ly01L, Ly01R;

Ly01:   L←AC1;                  Get index
        T←AC0, SH<0;            Get address, test for negative index
        MTEMP←L RSH 1, :Ly01P;  [Ly01P, Ly01N] Divide index by 2

Ly01N:  T←77777+T+1;            Negative index, extend sign of index/2
Ly01P:  MAR←MTEMP+T;                    Positive index, start fetch
        SINK←AC1, BUSODD; Which byte?
        T←377, :Ly01L;          [Ly01L, Ly01R]

Ly01L:  L←MD AND NOT T, TASK;  Left byte, mask and swap to right
        AC0←L LCY 8, :Bran2;

Ly01R:  L←MD AND T, TASK, :LtoAC0; Right byte, mask and store


; Load byte from array
; Loads the ac0'th byte from the array pointed to by ac1
; and returns it right-justified in ac1.
; Called by jsr @363
; Note:  ac0 may be negative.

!1,2, Ly10P, Ly10N;
!1,2, Ly10L, Ly10R;

Ly10:   L←AC0;                  Get index
        T←AC1, SH<0;            Get address, test for negative index
        MTEMP←L RSH 1, :Ly10P;  [Ly10P, Ly10N] Divide index by 2

Ly10N:  T←77777+T+1;            Negative index, extend sign of index/2
Ly10P:  MAR←MTEMP+T;                    Positive index, start fetch
        SINK←AC0, BUSODD; Which byte?
        T←377, :Ly10L;          [Ly10L, Ly10R]

Ly10L:  L←MD AND NOT T, TASK;  Left byte, mask and swap to right
        AC1←L LCY 8, :Bran2;

Ly10R:  L←MD AND T, TASK; Right byte, mask and store
        AC1←L, :Bran2;
```

```
; Store byte into array
; Stores the byte now contained in frame temp 3 (ac2!3) into
; the ac1'th byte of the array pointed to by ac0.
; Called by jsr@364
; Note:  ac1 may be negative.

!1,2, Sy01P, Sy01N;
!1,2, Sy01L, Sy01R;

Sy01:  L←AC1;               Get index
       T←3, SH<0;           Frame offset, test for negative index
       MAR←AC2+T, :Sy01P; [Sy01P, Sy01N] Start fetch of byte to store

Sy01N: MTEMP←L MRSH 1, :Sy01A; Negative index, divide by 2 and extend sign
Sy01P: MTEMP←L RSH 1;          Positive index, just divide by 2

Sy01A: T←MTEMP;             Get word index
       L←AC0+T;             Compute address of word
       T←MD;                Here comes the byte to store
       MTEMP←L;             Save word address
       MAR←MTEMP;                  Fetch word being stored into
       SINK←AC1, BUSODD; Which byte?
Sy01C: L←377 AND T, T←377, :Sy01L; [Sy01L, Sy01R] Isolate byte being stored

Sy01L: AC1←L LCY 8;         Storing into left byte, swap halves.
       L←MD AND T, :Sy01B;         Zero left byte of word being stored into

Sy01R: AC1←L;               Storing into right byte, already set up
       L←MD AND NOT T;             Zero right byte of word being stored into

Sy01B: MAR←MTEMP;                  Start store
       T←LREG;                     Existing contents to preserve
       L←AC1 OR T, TASK;    Merge old and new bytes
       MD←LREG, :Bran2;     Finish store, then PC←AC3 and go to START


; Store byte into array
; Stores the byte now contained in frame temp 3 (ac2!3) into
; the ac0'th byte of the array pointed to by ac1.
; Called by jsr@365
; Note:  ac0 may be negative.

!1,2, Sy10P, Sy10N;

Sy10:  L←AC0;               Get index
       T←3, SH<0;           Frame offset, test for negative index
       MAR←AC2+T, :Sy10P; [Sy10P, Sy10N] Start fetch of byte to store

Sy10N: MTEMP←L MRSH 1, :Sy10A; Negative index, divide by 2 and extend sign
Sy10P: MTEMP←L RSH 1;          Positive index, just divide by 2

Sy10A: T←MTEMP;             Get word index
       L←AC1+T;             Compute address of word
       T←MD;                Here comes the byte to store
       MTEMP←L;             Save word address
       MAR←MTEMP;                  Fetch word being stored into
       SINK←AC0, BUSODD, :Sy01C; Which byte? Join common code
```