* DRAFT *

Am1601

Programmer's Reference

Issue 1.0.12
October 29, 2002

This document is subject to modification.

The latest version of this document can be found here:

http://www.amsat.org/amsat/projects/ips/Am1601.html

AMSAT-BDA

This document is maintained by Paul C. L. Willmott, VP9MU.

Reports of errors should be sent to vp9mu@amsat.org

DOCUMENT HISTORY LOG

Status (Baseline/ Revision/ Canceled)	Document Revision	Effective Date	Description
Baseline	1.0.0	October 9, 2001	Initial Draft by Paul Willmott (PCLW) based upon notes by Lyle Johnson (LJ) March 22, 2001
Revision	1.0.1	August 17, 2002	Information on jumps and branches added by PCLW, based upon email from LJ August 10, 2002.
Revision	1.0.2	August 18, 2002	Overflow flag conditions and machine cycle info added to instruction descriptions by LJ. Compares and Stack Group instructions added. Corrections to RSC and PSC references in instruction descriptions, these grow up now. References to T and S replaced by P0 and P1 in all sections. Minor formatting edits. PCLW
Revision	1.0.3	August 24, 2002	Corrections and edits, CMP & TST P1 added, I & IE flags added by LJ. Minor formatting edits. GNU Free Documentation License stuff added. RCMP removed. Even addresses for 16 bit opcodes. Refresh Register, Error Counter and REFRESH, LSL, LSR, ASR, ROL, ROR, CPL (COM), BIT, CLREF, cLOADB, cSTOREB, cpLOADB, cpSTOREB, DI, EI, PUSHREF, POPREF, RTS, EXECUTE, EMULATE, PREPARE instructions added. Traps and Vectors placeholder added. I/O space details, IN & OUTB instructions added. Changes to cpStore description. icLOAD renamed cpLOAD. PCLW
Revision	1.0.4	August 25, 2002	Formatting tidy-up, not shown in markup. Consistency changes to instruction descriptions. Additions to BIT description. PCLW
Revision	1.0.5	August 27, 2002	Corrections, cIN and cOUTB changed to cpINB and cpOUTB by LJ. Formating changes. * DRAFT * Remove Before Flight added. PCLW.
Revision	1.0.6	August 31, 2002	Miscellaneous edits. Extra stuff on Interrupt/Reset. EA register added and support. Z flag set added to cc based instructions. sLOAD, sSTORE, uNLOAD,

			sNLOAD, cNLOAD added. STOR renamed PTOR to avoid confusion with STOREs. RTOS renamed to RTOP. Some examples added. PCLW
Revision	1.0.7	September 7, 2002	Corrections by LJ. Formatting tidy-up, not shown in markup. PUSHREF, POPREF opcode values changed. PUSHEA, POPEA added. Some examples added, some example placeholders removed. Minor language edits. Vectors for stack underflow added. PCLW
Revision	1.0.8	September 8, 2002	Corrections by LJ. Formatting tidy-up. Page numbering changes. cBSR, POPEA removed. PCLW
Revision	1.0.9	September 14, 2002	Misc corrections. EMULATE, PREPARE now conditional on External Flag. EXECUTE now conditional on Zero Flag. sBR added as #4x. sBSR changed to #5x. FLAG now 2 byte instruction and moved to #FF. uNLOAD, sNLOAD moved to #70, #71. PCLW Corrections by LJ. More corrections by PCLW. Assembler updated.
Revision	1.0.10	September 19, 2002	1 byte instructions changed to 2 byte instructions. PC odd vector added. EF in cc conditionals changed to not EF. BIT renamed as MASK. CLREF, DI, EI, POPREF, PUSHREF, POPEA, RTS deleted. P0/P1 variants of SBC/SUB/CMP reversed with RSBC/RSUB/RCMP. RCMP added back in. DFX, 2BLIT added. Opcode numbers changed everywhere. Corrections and additions by LJ and PCLW. PUSH & POP renamed PUSHPS & POPPS. PUSHRS & POPRS added. SET & CLEAR added. XB (SWAPB) added. EMULATE, EXECUTE. PREPARE & REFRESH now cc conditional. All 16 bit data structures must start at an even address. All instructions except the explicit byte manipulation instructions expect to find a word at an even address. Position of condition code cc changed in opcodes/operands. cIN, cOUT, cINB, cOUTB, cpIN, cpOUT added. cpINB/cpOUTB opcodes changed.
Revision	1.0.11	September 21, 2002	opcodes for subtracts switched around. FLAG is now a load rather than a push. Minor edits. LJ/PCLW.
Revision	1.0.12	October 29, 2002	Changes to FLAG condition codes, and cBR condition codes LJ. Copyright changed from AMSAT-BDA to AMSAT-DL. Minor edits. Am1601 Assembler and IPS-

			F1G source code added back in. PCLW.

CHAPTER 1 - Am1601 Overview

"The Am1601 is a stack based CPU implemented in a FPGA. Its reason for existence is to allow AMSAT access to an IPS-friendly radiation tolerant processor and to be in control of the intellectual property associated with it."

Lyle Johnson KK7P
March 22, 2001

Am1601 Emulator

An Am1601 emulator program for Windows NT, and a general user version of IPS to run on the emulator is available. The purpose of the emulator is to aid the design and development of the hardware Am1601, not to run "real-life" programs. IPS running on the emulator is considerably slower than it would be on a real hardware machine. That said the emulator is a very useful tool for debugging purposes.

It is made available to the general public for peer review purposes, and is provided "as is".

The emulator program can be downloaded from here:

http://www.amsat.org/amsat/projects/ips/Am1601.html

Additional IPS software and documentation can be found here:

http://www.amsat.org/amsat/sats/ao40/ips.html

CHAPTER 2 - Am1601 Architecture

2.1 Memory Space

The memory in an Am1601 system is a sequence of up to 65536 bytes. A *word* is any two consecutive bytes in memory. Words are stored in memory with the most significant byte at the higher address. Instructions that manipulate 16-bit values expect to find data in memory at even addresses. Only the explicit byte manipulation instructions are able to access data stored at odd addresses.

2.2 Stacks

The Am1601 processor has two internal last-in first-out (LIFO) stacks; the *parameter* and the *return.* The parameter stack is used to store data, whilst the return stack stores return addresses etc. These two internal stacks are 16 bits wide and 16 elements deep. These two internal stacks are implemented as bi-directional shift registers, e.g. a push operation onto the parameter stack will cause P14 to be shifted to P15, P13 to P14 etc. The Am1601 transparently handles stack overflow to external memory if necessary, with the only penalty being execution time.

All arithmetic and logical operations are performed on the contents of the parameter stack, with the results being deposited back onto the parameter stack.

Parameter Stack	
Top->	P0
Second->	P1
Third->	P2
	P3
	P4
	P5
	P6
	P7
	P8
	P9
	P10
	P11
	P12
	P13
	P14
	P15

Return Stack	
Top->	R0
	R1
	R2
	R3
	R4
	R5
	R6
	R7
	R8
	R9
	R10
	R11
	R12
	R13
	R14
	R15

2.3 Registers

In addition to the stack registers the Am1601 provides a *program counter (PC)* and a number of IPS-friendly support registers.

2.3.1 Program Counter (PC). The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented (by 2 or 4 depending upon the instruction being executed) after its contents have been transferred to the address lines. When a program jump occurs, the new value is automatically placed in the PC, overriding the incrementer. If the PC register is loaded with an odd numbered address, then the instruction being executed is aborted, the PC register is loaded with the address of the PC Odd Vector, and program execution continues at the address now indicated by the PC register.

2.3.2 Flag Register (FLAGS) supplies information to the user regarding the status of the Am1601 at any given time. The bit positions for each flag are shown below:

```
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
EC EC EC EC EC EC X  X  EE IE I  E  O  S  Z  C
b5 b4 b3 b2 b1 b0
```

Where:

C = Carry Flag
Z = Zero Flag
S = Sign Flag
O = Overflow Flag
E = External Flag
I = Interrupt Flag
IE = Interrupt Enable

EE = EDAC Error
EC = EDAC Error Counter
X = Not Used

The bits in the least significant byte of the FLAGS register can be set or cleared in software by the SET and CLEAR instructions. It is not recommended that the EE, E or I flags be set in software.

2.3.2.1 Carry Flag (C). The carry bit is set or reset depending on the operation being performed. For ADD instructions that generate a carry and SUBTRACT instructions that generate a borrow, the Carry Flag will be set. The Carry Flag is reset by an ADD that does not generate a carry and a SUBTRACT that doesn't generate a borrow. This saved carry facilitates software routines for extended precision arithmetic.

2.3.2.2 Zero Flag (Z). The Zero Flag is set or reset if the result generated by the execution of certain instructions is zero. For arithmetic and logical operations, the Z flag will be set to a 1 if the resulting word on the top of the parameter stack is zero. If the word is not zero, the Z flag is reset to 0. Certain conditionally executed instructions which may affect the parameter stack depth also set or clear the Z flag upon execution.

2.3.2.3 Sign Flag (S). The Sign Flag (S) stores the state of the most significant bit of the word on the top of the parameter stack (Bit 15). When the Am1601 performs arithmetic operations on signed numbers, binary two's complement notation is used to represent and process numeric information. A positive number is identified by a 0 in bit 15. A negative number is identified by a 1. The binary equivalent of the magnitude of a positive number is stored in bits 0 to 14 for a total range of 0 to 32767. A negative number is represented by the two's complement of the equivalent positive number. The total range for negative numbers is –1 to –32768.

2.3.2.4 Overflow Flag (O). The Overflow Flag (O) is valid for binary two's complement operations only (signed). The Overflow Flag is set to 1 if an overflow occurred, 0 otherwise. If bit 15 of both operands are set to 1, and bit 15 of the result is 0, then an overflow occurred. Likewise if bit 15 of both operands are set to 0, and bit 15 of the result is 1, then an overflow occurred.

2.3.2.5 External Flag (E). The External Flag is a flip-flop that is set by an external event, sampled at an I/O pin. It is cleared in software by the CLEAR instruction with the appropriate mask.

2.3.2.6 Interrupt Flag (I). The Interrupt Flag is set when a maskable interrupt occurs. It is cleared in software by the CLEAR instruction with the appropriate mask.

2.3.2.7 Interrupt Enable Flag (IE). The Interrupt Enable Flag is used to enable or disable the maskable interrupt. It is manipulated in software by the SET and CLEAR instructions with the appropriate mask. It is automatically cleared (disabled) in hardware when an Interrupt is responded to. It must subsequently be enabled by software.

2.3.2.8 EDAC Error Flag (EE). The EDAC Error Flag is set when the EDAC sub-system corrects a bit in memory. When an error is corrected, the EDAC Error Counter (EC) is incremented, and the EDAC Error Address register (EA) is loaded with the address of

the byte or word in memory corrected. It is cleared in software by the CLEAR instruction with the appropriate mask.

2.3.3 Pseudo Program Counter (PPC). The IPS pseudocode is executed by the inner interpreter. This routine employs a pointer, the so-called pseudo-program counter. This PPC points to the next pseudo-instruction to be executed in turn.

2.3.4 Header Pointer (HP). The IPS inner interpreter fetches the contents of the location that the PPC points to and then increments the PPC by two. This fetched number is called the header pointer (HP) and it points to the header of the routine to be executed.

2.3.5 Parameter Stack Pointer (PSP). The parameter stack pointer holds the address in memory of the top of the parameter stack overflow area. The overflow area is only used if the parameter stack contains more than 16 entries. When a value is pushed onto the overflow area, the Parameter Stack Pointer is post-decremented by 2. When a value is popped from the overflow area, the Parameter Stack Pointer is pre-incremented by 2. The Parameter Stack Pointer must be initialised to an even address by software.

2.3.6 Parameter Stack Counter (PSC). The parameter stack counter is an up/down counter that is incremented every time a push operation on the parameter stack occurs, and is decremented every time a pop operation on the parameter stack occurs. It allows the user to inspect the number of items on the parameter stack. It is also used to implement the parameter stack hardware overflow/underflow mechanism. This register is initialized to zero after a Reset. In the event of an underflow of the parameter stack (PSC decrements to #FFFF), the processor aborts the instruction currently being executed. Then the contents of the Program Counter are pushed onto the return stack. The PC register is then loaded with the address of the Parameter Stack Underflow Vector (#0008) and points to the next instruction to be executed.

2.3.7 Return Stack Pointer (RSP). The return stack pointer holds the address in memory of the top of the return stack overflow area. The overflow area is only used if the return stack contains more than 16 entries. When a value is pushed onto the overflow area, the Return Stack Pointer is post-decremented by 2. When a value is popped from the overflow area, the Return Stack Pointer is pre-incremented by 2. The Return Stack Pointer must be initialised to an even address by software.

2.3.8 Return Stack Counter (RSC). The return stack counter is an up/down counter that is incremented every time a push operation on the return stack occurs, and is decremented every time a pop operation on the return stack occurs. It allows the user to inspect the number of items on the return stack. It is also used to implement the return stack hardware overflow/underflow mechanism. This register is initialized to zero after a Reset. In the event of an underflow of the return stack (RSC decrements to #FFFF), the processor aborts the instruction currently being executed. Then the contents of the Program Counter are pushed onto the return stack. The PC register is then loaded with the address of the Return Stack Underflow Vector (#0004) and points to the next instruction to be executed.

2.3.9 Refresh Register (RR). The refresh register contains the address of the next memory location to be refreshed. This is used by the REFRESH instruction to implement the EDAC memory wash. The RR register is initialised to #0000 upon reset.

2.3.10 EDAC Error Address (EA). This register holds the address of the last byte or word in memory to be corrected by the EDAC sub-system. The EA register retains its contents through a reset.

2.4 Operand Notation

2.4.1 The following notation is used in the instruction descriptions:

1). P0 specifies the top register of the parameter stack.

2). (P0) specifies the contents of memory at the location addressed by the contents of the P0 register.

3). R0 specifies the top register of the return stack.

4). uN specifies a one byte value in the range 0 to 255.

5). sN specifies a one byte signed value in the range –128 to 127.

6). abc specifies a12 bit absolute address in the range #0000 to #0FFF. In the operand binary descriptions the “a” represents the high order nibble, and “c” the lowest order nibble.

7). efg specifies a 12 bit signed displacement in the range –2048 to 2047, bit 11 (high order bit) contains the sign. In the operand binary descriptions the “e” represents the high order nibble, and “g” the lowest order nibble.

8). cc specifies a condition code.

9). nnnn specifies a 16 bit value in the range 0 to 65535 (#0000 to #FFFF).

10). eeee specifies a 16 bit signed displacement in the range –32768 to 32767.

11). ee specifies an 8 bit signed displacement in the range –128 to 127.

12). s specifies any of the uN, sN or P1 operands. In the case of the SBC, SUB, and CMP instructions s specifies any of the uN, sN or P0 operands.

13). qq specifies any of the PC, PPC, HP, FLAGS, PSP, PSC, RSP, RSC, EA or RR registers.

14). Hexadecimal numbers are prefixed in this guide and the assembler by the # character, as per IPS convention.

15). <x> specifies a nibble (4 bits), where x can be any character. e.g. for a byte containing #FE, <F> is the high order nibble, and <E> the lower order nibble.

16). MSB means the Most Significant Byte of a word. LSB means the least significant byte of a word.

17). pppp specifies a 16 bit I/O port address in the range 0 to 65535 (#0000 to #FFFF).

18). b0, b1 ... b15 specify an individual bit; b0 specifies bit 0, the lowest significant bit.

19). y specifies either of the uN or P0 operands.

20). dest <- expr means that the expression expr is loaded into the destination dest. The destination can by a register, memory location, I/O port, or bit in any of the foregoing.

21). P0 <<- expr means that the expression expr is pushed onto the top of the parameter stack. Similarly, R0 <<- expr means that the exepression expr is pushed onto the top of the return stack.

22). dest <-> dest means that the contents of the two destinations are exchanged, i.e. swapped.

23). The enclosing of an expression wholly in parentheses indicates a memory address. The contents of the memory address equivalent to the expression value will be used as the operand value.

24). A binary literal value is represented by prefixing the value by "B" e.g. B0000000000000000.

25). m specifies a bit mask. Within the mask all bits are clear except the one identified by the value of m.

2.5 All instructions and all 16-bit data structures must be stored in a memory location with an even address.

2.6 Traps & Vectors

2.6.1 The following addresses are set aside for traps and vectors:

#0000 Reset
#0004 Return Stack Underflow
#0008 Parameter Stack Underflow
#000C PC Odd Vector
#0010 Maskable Interrupt

2.6.2 An external reset (RST) causes the processor to complete the instruction currently being executed. The RR, PSC and RSC registers are initialised to #0000. All other registers are remain unchanged. Then the Program Counter (PC) is then loaded with the address of the Reset Vector (#0000) and points to the next instruction to be executed.

2.6.3 An external interrupt (INT) causes the maskable interrupt to be disabled. Then the processor completes the instruction currently being executed. Then the contents of the Program Counter are pushed onto the return stack. The PC register is then loaded with the address of the Maskable Interrupt Vector (#0010) and points to the next instruction to be executed.

2.6.4 In the event of an underflow of the return stack (RSC decrements to #FFFF), the processor aborts the instruction currently being executed. Then the contents of the Program Counter are pushed onto the return stack. The PC register is then loaded with the address of the Return Stack Underflow Vector (#0004) and points to the next instruction to be executed.

2.6.5 In the event of an underflow of the parameter stack (PSC decrements to #FFFF), the processor aborts the instruction currently being executed. Then the contents of the Program Counter are pushed onto the return stack. The PC register is then loaded with the address of the Parameter Stack Underflow Vector (#0008) and points to the next instruction to be executed.

2.6.6 In the event of the Program Counter Register (PC) being loaded with an odd value, then the current instruction being executed is aborted. The PC register is loaded with the PC Odd Vector (#000C), and program execution continues at the location indicated by the new contents of the PC register.

2.7 Input / Output (I/O)

2.7.1 The Am1601 has a separate I/O space from memory. This space is a bank of 65536 I/O ports addressed as #0000 to #FFFF. The ports are 8 bits wide. 16 bit ports can be accommodated at even I/O port addresses. The cIN, cOUT, cINB, cOUTB, cpIN, cpOUT, cpINB and cpOUTB instructions are used to read from and write to the I/O ports.

2.8 Undefined Opcodes

2.8.1 The opcode values for which instructions have not been defined, are just that ..."undefined and unpredictable".

CHAPTER 3 - Am1601 Instruction Set

INTRODUCTION:

This chapter describes each Am1601 opcode (instruction) in detail. The opcodes are largely presented in alphabetical order, one per page. Each instruction is introduced by its mnemonic opcode and symbolic operations. Then follows a brief description, operation, valid operand combinations, machine code, detailed description, condition bits affected, and one or more examples.

2BLIT

Operation: P0 <<- (PPC)
PSC <- PSC + 1
PPC <- PPC + 2
PC <- PC + 2

Format:

2BLIT

#FB

1	1	1	1	1	1	0	1

#00

0	0	0	0	0	0	0	0

Description:

This instruction fetches a 16-bit word at the address indicated by the Pseudo Program Counter Register (PPC) and pushes it onto the top of the parameter stack (P0). The PPC register is incremented by 2, and program execution continues at the next instruction in memory (PC + 2).

M CYCLES: 4

Condition Bits Affected:

None

ADC

Operation: if operand is uN or sN
P0 <- P0 + s + C
PC <- PC + 2

if operand is P1
P0 <- P0 + P1 + C
PSC <- PSC – 1
PC <- PC + 2

Format:

[uN|sN] s ADC

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN ADC

#A1

1	0	1	0	0	0	0	1

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN ADC

#B1

1	0	1	1	0	0	0	1

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 ADC

#C0

1	1	0	0	0	0	0	0

#10

0	0	0	1	0	0	0	0

Description:

The byte or word specified by the s operand, along with the Carry Flag (“C” in the Flags register) are added to the contents of the top register of the parameter stack (P0); the result replaces the contents of P0. In the case of the P1 variant, the top two entries on the parameter stack are popped and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Set if carry from Bit 15; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Set if the signed result is an overflow; reset otherwise

Example:

If the P0 register contains #0016, the Carry Flag is set, the P1 register contains #0010, after the execution of

P1 ADC

the P0 register will contain #0027.

ADD

Operation: if operand is uN or sN
P0 <- P0 + s
PC <- PC + 2

if operand P1
P0 <- P0 + P1
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s ADD

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN ADD

#A0

1	0	1	0	0	0	0	0

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN ADD

#B0

1	0	1	1	0	0	0	0

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 ADD

#C0

1	1	0	0	0	0	0	0

#00

0	0	0	0	0	0	0	0

Description:

The byte or word specified by the s operand is added to the contents of the top register of the parameter stack (P0); the result replaces the contents of P0. In the case of the P1 variant, the top two entries on the parameter stack are popped and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Set if carry from Bit 15; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Set if the signed result is an overflow; reset otherwise

Example:

If the contents of the P0 register are #00A0, and the uN operand has the value #02, after the execution of

```
#02 uN ADD
```

the P0 register will contain #00A2.

AND

Operation: if operand is uN or sN
P0 <- P0 AND s
PC <- PC + 2

if operand is P1
P0 <- P0 AND P1
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s AND

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN AND

#A6

1	0	1	0	0	1	1	0

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN AND

#B6

1	0	1	1	0	1	1	0

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 AND

#C0

1	1	0	0	0	0	0	0

#60

0	1	1	0	0	0	0	0

Description:

A logical AND operation, bit by bit, is performed between the byte or word specified by the s operand and the contents of the top register of the parameter stack (P0); the result replaces the contents of P0. In the case of the P1 variant, the top two entries on the parameter stack are popped and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Reset

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset.

Example:

If the P0 register contains #007B (B0000000001111011) and the P1 register contains #00C3 (B0000000011000011) after the execution of

P1 AND

the P0 register will contain #0043 (B0000000001000011).

ASR

Operation: P0 <- P0, C<b0, b0<b1, b1<-b2 .. b14<b15, b15<b15
PC <- PC + 2

Format:

ASR

#C2

1	1	0	0	0	0	1	0

#90

1	0	0	1	0	0	0	0

Description:

An arithmetic shift right is performed on the contents of the top register of the parameter stack (P0). The content of bit 15 (b15) is copied into bit 14 (b14); the previous content of bit 14 is copied into bit 13; this pattern is continued throughout the word. The content of bit 0 (b0) is copied into the Carry Flag ("C" in the FLAGS register), and the previous content of bit 15 (b15) is unchanged. Bit 0 (b0) s the least significant bit.

M CYCLES: 2

Condition Bits Affected:

C: Data from Bit 0 of previous contents of P0

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset.

Example:

If the P0 register contains #8038 (B1000000000111000) after the execution of

ASR

the contents of the P0 register will be #C01C (B1100000000011100) and the Carry Flag will contain 0.

cBR

Operation: if condition cc = 0
PC <- PC + 2

if condition cc = 1
PC <- PC + ee + 2

Format:

The cc operand is any of the condition codes as defined for the FLAG instruction. ee is the destination displacement.

ee cc cBR

#9<cc>

1	0	0	1	<	c	c	>

ee

<			e	e			>

Description:

This instruction provides for conditional branching. If the condition cc is met then the destination displacement (ee) is added to the Program Counter (PC) + 2 and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode + 2 and has a range of –128 to 127. If the condition is not met then the address of the next instruction in memory (PC + 2) is loaded into the PC register and points to the next program instruction to be executed.

The assembler provides the following definitions to aid the computation of jump offsets:

cc cBRbegin	Stores opcode and leaves address of the place to insert the jump offset on top of the IPS-X parameter stack.
cBRelse	Calculates and inserts the required jump offset into the address previously deposited by cBRbegin on the top of the IPS-X parameter stack. The address is popped and discarded. Then an AL cBR is assembled into the code, and the address of the place to insert the jump offset is pushed onto the top of the IPS-X parameter stack.

cBRend	Calculates and inserts the required jump offset into the address previously deposited by cBRbegin or cBRelse. The address is popped and discarded.

M CYCLES: 3 if branch taken; 2 if branch not taken.

Condition Bits Affected:

None

Example:

To jump forward 6 locations from address #0480 if the Z flag is set, the following assembly language statement is used:

```
4 EQ cBR
```

The resulting object code and final PC value is shown below:

Location	Contents
480	#90
481	#04
482	
483	
484	
485	
486	<- PC after jump

sBR

Operation: PC <- PC + efg + 2

Format:

efg sBR

#4<e>

0	1	0	0	<	e		>

<f><g>

<	f		>	<	g		>

Description:

This instruction provides for unconditional branching. The value of the displacement efg is added to the contents of the Program Counter (PC) + 2 and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode + 2 and has a range of –2048 to 2047.

The assembler provides the following definitions to aid the computation of jump offsets:

sBRbegin	Stores opcode and leaves address of the place to insert the jump offset on top of the IPS-X parameter stack.
sBRcomplete	Calculates and inserts the required jump offset into the address previously deposited by sBRbegin. The address is popped and discarded.

M CYCLES: 3

Condition Bits Affected:

None

Example:

To jump to the address #0108, the following assembly language statement is used:

#006 sBR

The resulting object code and final PC value is shown below:

Location	Contents
100	#40
101	#06
102	
103	
104	
105	
106	
107	
108	<- PC after jump

cpBSR

Operation: if condition cc = 0
PSC <- PSC - 1
PC <- PC + 2

if condition cc = 1
R0 <- PC + 2
RSC <- RSC + 1
PC <- PC + P0 + 4
PSC <- PSC - 1

Format:

cc cpBSR

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpBSR

#8F

1	0	0	0	1	1	1	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

This instruction provides for conditional branching to a subroutine. If the condition cc is met then the address of the next instruction in memory (PC + 2) is pushed onto the return stack. The displacement value is popped off the parameter stack and added to the contents of the Program Counter (PC) + 4 and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode + 4 and has a range of –32768 to 32767. If the condition is not met then the top of the parameter stack is popped and the address of the next instruction in memory (PC + 2) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 3 if branch taken; 2 if branch not taken.

Condition Bits Affected:

None

Example:

If the top register of the parameter stack (P0) contains 4. To jump to a subroutine located at the displacement indicated by the P0 register if the Z flag is set, the following assembly language statement is used:

EQ cpBSR

The resulting object code and final PC value is shown below:

Location	Contents
480	#8F
481	#00
482	
483	
484	
485	
486	<- PC after jump, R0 <<- PC + 2

sBSR

Operation: R0 <- PC + 2
RSC <- RSC + 1
PC <- PC + efg + 2

Format:

efg sBSR

#5<e>

0	1	0	1	<	e		>

<f><g>

<	f		>	<	g		>

Description:

This instruction provides for unconditional branching to a subroutine. The address of the next instruction to be executed is pushed onto the return stack. Then the value of the displacement efg is added to the contents of the Program Counter (PC) + 2 and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode + 2 and has a range of –2048 to 2047.

cc sBSRbegin	Stores opcode and leaves address of the place to insert the jump offset on top of the IPS-X parameter stack.
sBSRcomplete	Calculates and inserts the required jump offset into the address previously deposited by sBSRbegin. The address is popped and discarded.

M CYCLES: 3

Condition Bits Affected:

None

Example:

To jump to a subroutine located at address #0108, the following assembly language statement is used:

#008 sBSR

The resulting object code and final PC value is shown below:

Location	Contents
100	#50
101	#06
102	
103	
104	
105	
106	
107	
108	<- PC after jump, R0 <<- PC + 2

CLEAR

Operation: FLAGS <<- FLAGS AND NOT MASK(m)
PC <- PC + 2

Format:

m CLEAR

#D5

1	1	0	1	0	1	0	1

#<m>0

<	m		>	0	0	0	0

Description:

The bit corresponding to the value of m in the FLAGS register is cleared according to the following table:

FLAGS bit	m	
C	0000	#00
Z	0001	#01
S	0010	#02
O	0011	#03
E	0100	#04
I	0101	#05
IE	0110	#06
EE	0111	#07

M CYCLES: 2

Condition Bits Affected:

The bit indicated by the m operand is cleared.

CMP

Operation: if operand is uN or sN
P0 - s
PC <- PC + 2

if operand is P0
P1 – P0
PC <- PC + 2

Format:

[uN|sN] s CMP

The s operand is any of uN , sN or P0. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN CMP

#AB

1	0	1	0	1	0	1	1

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN CMP

#BB

1	0	1	1	1	0	1	1

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P0 CMP

#C0

1	1	0	0	0	0	0	0

#A0

1	0	1	0	0	0	0	0

Description:

In the case of the sN or uN variants the word specified by the s operand is compared to (subtracted from) the contents of the top register of the parameter stack (P0) and the condition flags are set. In the case of the P0 variant the top register of the parameter stack (P0) is compared to (subtracted from) the second register of the parameter stack (P1) and the condition flags are set. The contents of the parameter stack remain unchanged.

M CYCLES: 2

Condition Bits Affected:

C: Set if there was a borrow; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

0: Set if signed overflow; reset otherwise

Example:

If the P0 register contains #FF45 and the P1 register contains #FF45, then after the execution of

P0 CMP

the Zero Flag (Z) will be set

CPL

Operation: P0 <- 1 - P0
PC <- PC + 2

Format:

CPL

#C0

1	1	0	0	0	0	0	0

#D0

1	1	0	1	0	0	0	0

Description:

The contents of the top register of the parameter stack (P0) are inverted (1's complement). This is the same as subtracting the contents of the P0 register from 1.

M CYCLES: 2

Condition Bits Affected:

C: Set if P0 was not #0001 before operation; reset otherwise.

S: Set if result is negative; reset otherwise.

Z: Set if result is 0; reset otherwise.

O: Reset.

Example:

If the contents of the P0 register are #00B4 (B0000000010110100), after the execution of

CPL

the P0 register will be #FF4B (B1111111101001011).

DEL

Operation: PSC <- PSC - 1
PC <- PC + 2

Format:

DEL

#C1

1	1	0	0	0	0	0	1

#10

0	0	0	1	0	0	0	0

Description:

The top entry of the parameter stack is popped and discarded.

M CYCLES: 2

Condition Bits Affected:

None

Example:

	P2	P1	P0
Before operation:	8	5	12
After operation:	-	8	5

DFX

Operation: R0 <<- PPC
RSC <- RSC + 1
PPC <- HP
PC <- PC + 2

Format:

DFX

#F8

1	1	1	1	1	0	0	0

#00

0	0	0	0	0	0	0	0

Description:

Definition Executive: the contents of the Pseudo Program Counter Register (PPC) are pushed onto the top of the return stack (R0), then the contents of the PPC register are replaced by the contents of the Header Pointer Register (HP). Program execution continues at the next instruction in memory (PC + 2).

M CYCLES: 3

Condition Bits Affected:

None

DUPL

Operation: P0 <<- P0
PSC <- PSC + 1
PC <- PC + 2

Format:

DUPL

#C1

1	1	0	0	0	0	0	1

#00

0	0	0	0	0	0	0	0

Description:

The contents of the top register of the parameter stack (P0) are pushed onto the top of the stack, i.e. duplicated.

M CYCLES: 2

Condition Bits Affected:

None

Example:

	P2	P1	P0
Before operation:		5	12
After operation:	5	12	12

EOR

Operation: if operand is uN or sN
P0 <- P0 EOR s
PC <- PC + 2

if operand is P1
P0 <- P0 EOR P1
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s EOR

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN EOR

#A8

1	0	1	0	1	0	0	0

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN EOR

#B8

1	0	1	1	1	0	0	0

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 EOR

#C0

1	1	0	0	0	0	0	0

#80

1	0	0	0	0	0	0	0

Description:

A logical exclusive-OR operation, bit by bit, is performed between the byte or word specified by the s operand and the contents of the top register of the parameter stack (P0); the result replaces the contents of P0. In the case of the P1 variant, the top two entries on the parameter stack are popped and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Reset

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset

Example:

If the P0 register contains #0096 (B0000000010010110), and the operand uN has the value #5D (B01011101), then after the execution of

#0096 uN EOR

the P0 register will contain #00CB (B0000000011001011).

EMULATE (not available in prototype)

Operation:
```
if condition cc = 1
        HP <- (PPC)
        PPC <- PPC + 2
        PC <- (HP)
        HP <- HP + 2
if condition cc = 0
        PC <- PC + 2
```

Format:

cc EMULATE

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc EMULATE

#F0

1	1	1	1	0	0	0	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met, then EMULATE loads the Header Pointer Register (HP) with the contents of the word in memory at the address indicated by the Pseudo Program Counter Register (PPC). The PPC register is incremented by 2. The Program Counter (PC) is then loaded with the address stored at the location indicated by the new contents of the HP register. The HP register is then incremented by 2. Program execution continues from the location now stored in the PC register. If the condition cc is not met then program execution continues at the next instruction (PC + 2).

M CYCLES: 5 cycles if cc, 2 cycles if not cc

Condition Bits Affected:

None

EXECUTE

Operation: if condition cc = 1
PC <- (HP)
HP <- HP + 2
if condition cc = 0
PC <- PC + 2

Format:

cc EXECUTE

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc EXECUTE

#F1

1	1	1	1	0	0	0	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the Program Counter (PC) is loaded with the address stored at the location indicated by the contents of the Header Pointer Register (HP). The HP register is then incremented by 2. Program execution continues from the location now stored in the PC register. If the condition cc is not met then program execution continues at the next instruction in memory (PC + 2).

M CYCLES: 3 cycles if cc; 2 cycles if not cc

Condition Bits Affected:

None

FLAG

Operation: if condition cc = 0
P0 <<- #0000
PC <- PC + 2

if condition cc = 1
P0 <<-#0001
PC <- PC + 2

Format:

cc FLAG

#FF

1	1	1	1	1	1	1	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met, then #0001 replaces the contents of the top register of the parameter stack (P0). If the condition is not met then #0000 replaces the contents of the P0 register.

Condition Codes

#0	EQ	Equal, Zero Flag (Z) = 1
#1	NE	Not Equal, Zero Flag (Z) = 0
#2	CS/HI	Carry Set/Higher, Carry Flag (C) = 1
#3	CC/LS	Carry Clear/Lower or Same, Carry Flag (C) = 0
#4	MI	Minus, Sign Flag (S) = 1
#5	PL	Plus, Sign Flag (S) = 0
#6	VS	Overflow Flag (O) = 1
#7	VC	Overflow Flag (O) = 0

#8	HS	Higher or Same, Zero Flag (Z) = 1 or Carry Flag (C) = 1
#9	LO	Lower, Zero Flag (Z) = 0 and Carry Flag (C) = 0
#A	GE	Greater Than or Equal, (Sign Flag (S) = 1 and Overflow Flag (O) = 1) or (Sign Flag (S) = 0 and Overflow Flag (O) = 0)
#B	LT	Less Than, (Sign Flag (S) = 1 and Overflow Flag (O) = 0) or (Sign Flag (S) = 0 and Overflow Flag (O) = 1)
#C	GT	Greater Than, Zero Flag (Z) = 0 and ((Sign Flag (S) = 1 and Overflow Flag (O) = 1) or (Sign Flag (S) = 0 and Overflow Flag (O) = 0))
#D	LE	Less Than or Equal, Zero Flag (Z) = 1 or ((Sign Flag (S) = 1 and Overflow Flag (O) = 0) or (Sign Flag (S) = 0 and Overflow Flag (O) = 1))
#E	AL	Always (true)
#F	NEF	External Flag (E) = 0. The External Flag (E) is a flip flop that can be set by an external event, this is intended to support the IPS “pseudo interrupt”.

M CYCLES: 2

Condition Bits Affected:

None

Example:

If the Zero Flag (Z) contains 1, then after the execution of

```
EQ FLAG
```

the P0 register will contain #0001.

IDX

Operation: P0 <<- R0
PSC <- PSC + 1
PC <- PC + 2

Format:

IDX

#C1

1	1	0	0	0	0	0	1

#80

1	0	0	0	0	0	0	0

Description:

The contents of the top register of the return stack (R0) are pushed onto the top of the parameter stack. The return stack remains unchanged.

M CYCLES: 2

Condition Bits Affected:

None

cIN

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
P0 <<- (nnnn)
Z <- 1
PSC <- PSC + 1
PC <- PC + 4

Format:

nnnn cc cIN

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the address of the load value.

nnnn cc cIN

#E2

1	1	1	0	0	0	1	0

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the value of the input port at the I/O address indicated by the operand nnnn is pushed onto the top of the parameter stack. Execution continues at the instruction following the operand nnnn in memory. If the condition is not met then the

address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cINB

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
P0 <<- (nnnn)
Z <- 1
PSC <- PSC + 1
PC <- PC + 4

Format:

nnnn cc cINB

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the address of the load value.

nnnn cc cINB

#E4

1	1	1	0	0	1	0	0

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the value of the input port at the I/O address indicated by the operand nnnn is pushed into the least significant byte of the P0 register. The most significant byte of P0 is filled with zeroes. Execution continues at the instruction following

the operand nnnn in memory. If the condition is not met then the address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cpIN

Operation: if condition cc = 0
Z <- 0
PSC <- PSC - 1
PC <- PC + 2

if condition cc = 1
P0 <- (P0)
Z <- 1
PC <- PC + 2

Format:

cc cpIN

The cc operand is any of the condition codes as defined for the FLAG instruction

cc cpIN

#EA

1	1	1	0	1	0	1	0

#0<cc>

0	0	0	0	<	C	c	>

Description:

If the condition cc is met then the value of the input port at the I/O address indicated by the contents of the top register of the parameter stack (P0) is loaded into the P0 register. If the condition is not met, then the top entry of the parameter stack is popped and discarded.

M CYCLES: 3 if condition cc is met; 2 otherwise.

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cpINB

Operation: if condition cc = 0
Z <- 0
PSC <- PSC - 1
PC <- PC + 2

if condition cc = 1
P0 <- (P0)
Z <- 1
PC <- PC + 2

Format:

cc cpINB

The cc operand is any of the condition codes as defined for the FLAG instruction

cc cpINB

#EC

1	1	1	0	1	1	0	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the value of the input port at the I/O address indicated by the contents of the top register of the parameter stack (P0) is loaded into the least significant byte of the P0 register. The most significant byte of P0 is filled with zeroes. If the condition is not met, then the top entry of the parameter stack is popped and discarded.

M CYCLES: 3 if condition cc is met; 2 otherwise.

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cJMP

Operation: if condition cc = 0
PC <- PC + 4

if condition cc = 1
PC <- nnnn

Format:

nnnn cc cJMP

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the destination address.

nnnn cc cJMP

#81

1	0	0	0	0	0	0	1

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the destination address nnnn is loaded into the Program Counter Register (PC) and points to the address of the next program instruction to be executed. If the condition is not met then the address of the instruction following the destination address in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed.

The assembler provides the following definitions to aid the computation of jump addresses:

cc cJMPbegin	Stores opcode and leaves address of the place to insert the jump address on top of the IPS-X parameter stack.
cJMPelse	Calculates and inserts the required jump address into the address previously deposited by cJMPbegin on the top of the IPS-X parameter stack. The address is popped and discarded. Then an AL cJMP is assembled into the code, and the address of the place to insert the jump address is pushed onto the top of the IPS-X parameter stack.
cJMPend	Calculates and inserts the required jump address into the address previously deposited by cJMPbegin or cJMPelse. The address is popped and discarded.

M CYCLES: 3

Condition Bits Affected:

None

cpJMP

Operation: if condition cc = 0
PSC <- PSC - 1
PC <- PC + 2

if condition cc = 1
PC <- P0
PSC <- PSC - 1

Format:

cc cpJMP

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpJMP

#89

1	0	0	0	1	0	0	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the top of the parameter stack is popped into the Program Counter Register (PC) and points to the address of the next program instruction to be executed. If the condition is not met then the top entry of the parameter stack is popped and the address of the next instruction in memory (PC + 2) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 2

Condition Bits Affected:

None

sJMP

Operation: PC <- abc

Format:

abc sJMP

#0<a>

0	0	0	0	<	a		>

<b><c>

<	b		>	<	c		>

Description:

Unconditional jump to a 12-bit address specified by abc. The operand abc is loaded into the Program Counter Register (PC) and points to the address of the next program instruction to be executed.

The assembler provides the following definitions to aid the computation of jump addresses:

sJMPbegin	Stores opcode and leaves address of the place to insert the jump address on top of the IPS-X parameter stack.
sJMPcomplete	Calculates and inserts the required jump address into the address previously deposited by sJMPbegin. The address is popped and discarded.

M CYCLES: 2

Condition Bits Affected:

None

JPPC

Operation: PPC <<- (PPC)
PC <- PC + 2

Format:

JPPC

#FC

1	1	1	1	1	1	0	0

#00

0	0	0	0	0	0	0	0

Description:

The contents of the word in memory at the address indicated by the contents of the Pseudo Program Counter Register (PPC) are loaded into the PPC register. Program execution continues at the next instruction in memory (PC + 2).

M CYCLES: 3

Condition Bits Affected:

None

cJSR

Operation: if condition CC = 0
PC <- PC + 4

if condition cc = 1
R0 <- PC + 4
RSC <- RSC + 1
PC <- nnnn

Format:

nnnn cc cJSR

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the address of a subroutine.

nnnn cc cJSR

#80

1	0	0	0	0	0	0	0

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the address of the instruction following the subroutine address in memory (PC + 4) is pushed onto the top of the return stack. The subroutine address nnnn is loaded into the Program Counter Register (PC) and points to the address of the next program instruction to be executed. If the condition is not met then the address of the instruction following the subroutine address in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed.

The assembler provides the following definitions to aid the computation of jump addresses:

cc cJSRbegin	Stores opcode and leaves address of the place to insert the jump address on top of the IPS-X parameter stack.
cJSRcomplete	Calculates and inserts the required jump address into the address previously deposited by cJSRbegin. The address is popped and discarded.

M CYCLES: 3

Condition Bits Affected:

None

cpJSR

Operation: if condition cc = 0
PSC <- PSC - 1
PC <- PC + 2

if condition cc = 1
R0 <<- PC + 2
RSC <- RSC + 1
PC <- P0
PSC <- PSC - 1

Format:

cc cpJSR

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpJSR

#88

1	0	0	0	1	0	0	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the address of the next instruction in memory (PC + 2) is pushed onto the return stack. Top of the parameter stack is popped into the Program Counter Register (PC) and points to the address of the next program instruction to be executed. If the condition is not met then the top of the parameter stack is popped and the address of the next instruction in memory (PC + 2) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 3

Condition Bits Affected:

None

sJSR

Operation: R0 <<- PC + 2
RSC <- RSC + 1
PC <- abc

Format:

abc sJSR

#1<a>

0	0	0	1	<	a		>

<b><c>

<	b		>	<	c		>

Description:

Unconditional jump to a subroutine whose 12-bit address is specified by abc. The address of the next instruction in memory (PC + 2) is pushed onto the return stack, then the operand abc is loaded into the Program Counter Register (PC) and points to the address of the next program instruction to be executed.

The assembler provides the following definitions to aid the computation of jump addresses:

sJSRbegin	Stores opcode and leaves address of the place to insert the jump address on top of the IPS-X parameter stack.
sJSRcomplete	Calculates and inserts the required jump address into the address previously deposited by sJSRbegin. The address is popped and discarded.

M CYCLES: 3

Condition Bits Affected:

None

cLOAD

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
P0 <<- (nnnn)
Z <- 1
PSC <- PSC + 1
PC <- PC + 4

Format:

nnnn cc cLOAD

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the address of the load value.

nnnn cc cLOAD

#82

1	0	0	0	0	0	1	0

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the word located at memory address indicated by the operand nnnn is pushed onto the top of the parameter stack. Execution continues at the instruction following the operand nnnn in memory. If the condition is not met then the

address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cLOADB

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
P0 <<- (nnnn)
Z <- 1
PSC <- PSC + 1
PC <- PC + 4

Format:

nnnn cc cLOADB

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the address of the load value.

nnnn cc cLOADB

#84

1	0	0	0	0	1	0	0

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the byte located at memory address indicated by the operand nnnn is pushed onto the top of the parameter stack. The byte is placed into the least significant byte of the P0 register; the upper byte being filled with zeroes.

Execution continues at the instruction following the operand nnnn in memory. If the condition is not met then the address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cNLOAD

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
P0 <<- nnnn
Z <- 1
PSC <- PSC + 1
PC <- PC + 4

Format:

nnnn cc cNLOAD

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the address of the load value.

nnnn cc cNLOAD

#60

0	1	1	0	0	0	0	0

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the word operand nnnn is pushed onto the top of the parameter stack. Execution continues at the instruction following the operand nnnn in memory. If the condition is not met then the address of the instruction following the

operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 3 if condition cc is met; 2 otherwise.

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cpLOAD

Operation: if condition cc = 0
Z <- 0
PSC = PSC - 1
PC <- PC + 2

if condition cc = 1
P0 <- (P0)
Z <- 1
PC <- PC + 2

Format:

cc cpLOAD

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpLOAD

#8A

1	0	0	0	1	0	1	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

The top entry of the parameter stack is popped. If the condition cc is met then the value in memory at the address pointed to by the address in the previous top of the parameter stack is pushed onto the top of the parameter stack (P0).

M CYCLES: 3 if condition cc is met; 2 otherwise.

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cpLOADB

Operation: if condition cc = 0
Z <- 0
PSC = PSC - 1
PC <- PC + 2

if condition cc = 1
P0 <- LSB (P0)
Z <- 1
PC <- PC + 2

Format:

cc cpLOADB

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpLOADB

#8C

1	0	0	0	1	1	0	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

The top entry of the parameter stack is popped. If the condition cc is met then the value of the byte in memory at the address pointed to by the address in the previous top of the parameter stack is pushed onto the top of the parameter stack (P0). The byte is placed in the least significant byte of the P0 register. The most significant byte of the P0 register is filled with zeroes.

M CYCLES: 3 if condition cc is met; 2 otherwise.

Condition Bits Affected:

Z Set if condition met, reset otherwise.

sLOAD

Operation: P0 <- (abc)
PC <- PC + 2

Format:

abc sLOAD

#2<a>

0	0	1	0	<	a		>

<b><c>

<	b		>	<	c		>

Description:

The contents of the word in memory at the address specified by the 12 bit operand is loaded into the top of the parameter stack (P0).

M CYCLES: 3

Condition Bits Affected:

None

sNLOAD

Operation: P0 <<- sN
PSC <- PSC + 1
PC <- PC + 2

Format:

sN sNLOAD

#71

0	1	1	1	0	0	0	1

sN

<			s	N			>

Description:

This instruction is an unconditional push of the signed operand sN onto the top of the parameter stack. The operand sN is placed in the least significant byte of the top register of the parameter stack (P0), the most significant byte of the P0 register is filled with the most significant bit of the operand.

M CYCLES: 2

Condition Bits Affected:

None

uNLOAD

Operation: P0 <<- uN
PSC <- PSC + 1
PC <- PC + 2

Format:

uN uNLOAD

#70

0	1	1	1	0	0	0	0

uN

<			u	N			>

Description:

This instruction is an unconditional push of the unsigned operand uN onto the top of the parameter stack. The operand uN is placed in the least significant byte of the top register of the parameter stack (P0), the most significant byte of the P0 register is filled with zeroes.

M CYCLES: 2

Condition Bits Affected:

None

LSL

Operation: P0 <- P0, C<b15, b15<b14, b14<-b13 .. b0<0

PC <- PC + 2

Format:

LSL

#C2

1	1	0	0	0	0	1	0

#00

0	0	0	0	0	0	0	0

Description:

An logic shift left is performed on the contents of the top register of the parameter stack (P0). Bit 0 (b0) is reset, the previous content of bit 0 (b0) is copied into bit 1 (b1); this pattern is continued throughout the word. The content of bit 15 (b15) is copied into the Carry Flag ("C" in the FLAGS register). Bit 0 is the least significant bit.

M CYCLES: 2

Condition Bits Affected:

C: Data from Bit 15 of previous contents of P0

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset.

LSR

Operation: P0 <- P0, C<b0, b0<b1, b1<-b2 .. b15<0
PC <- PC + 2

Format:

LSR

#C2

1	1	0	0	0	0	1	0

#10

0	0	0	1	0	0	0	0

Description:

An logical shift right is performed on the contents of the top register of the parameter stack (P0). Bit 15 (b15) is reset, the previous content of bit 15 (b15) is copied into bit 14 (b14); this pattern is continued throughout the word. The content of bit 0 (b0) is copied into the Carry Flag ("C" in the FLAGS register). Bit 0 is the least significant bit.

M CYCLES: 2

Condition Bits Affected:

C: Data from Bit 0 of previous contents of P0

S: Reset

Z: Set if result is zero; reset otherwise

O: Reset.

MASK

Operation: if operand is uN

P0 <- bit specified by low-order four bits of the operand uN is set, all other bits are cleared.
PC <- PC + 2

if operand is P0

P0 <- bit specified by low-order four bits of P0 is set, all other bits are cleared.
PC <- PC + 2

Format:

[uN] y MASK

The y operand is either of uN or P0. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN MASK

#AC

1	1	0	0	1	1	0	0

8-bit unsigned value, only the low-order four bits are used.

<			u	N			>

P0 MASK

#C0

1	1	0	0	0	0	0	0

#C0

1	1	0	0	0	0	0	0

Description:

The top entry of the parameter stack (P0) is popped. A mask is pushed onto the top of the parameter stack. The mask has all bits cleared except for the bit specified by the low order four bits of the s operand, which is set.

M CYCLES: 2

Condition Bits Affected:

C: Reset

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset

Example:

If the the P0 register contains #0002 after the execution of

```
P0 BIT
```

the P0 register will contain #0004 (B0000000000000100).

NEG

Operation: P0 <- 0 – P0
PC <- PC + 2

Format:

NEG

#C0

1	1	0	0	0	0	0	0

#F0

1	1	1	1	0	0	0	0

Description:

The contents of the top of the parameter stack (P0) are negated (two's complement). This is the same as subtracting the contents of the P0 register from zero. Note that #8000 is left unchanged.

M CYCLES: 2

Condition Bits Affected:

C: Set if P0 was not #0000 before operation; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset

Example:

If the top of the Parameter Stack contains:

#0001

after the execution of

NEG

The top of the parameter stack will contain:

#FFFF

NOP

Operation: PC <- PC + 2

Format:

NOP

#C0

1	1	0	0	0	0	0	0

#90

1	0	0	1	0	0	0	0

Description:

The processor performs no operation during this machine cycle.

M CYCLES: 2

Condition Bits Affected:

None

OR

Operation: if operand is uN or sN
P0 <- P0 OR s
PC <- PC + 2

if operand is P1
P0 <- P0 OR P1
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s OR

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN OR

#A7

1	0	1	0	0	1	1	1

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN OR

#B7

1	0	1	1	0	1	1	1

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 OR

#C0

1	1	0	0	0	0	0	0

#70

0	1	1	1	0	0	0	0

Description:

A logical OR operation is performed, bit by bit, between the byte or word specified by the s operand and the contents of the top register of the parameter stack (P0); the result replaces the contents of the P0 register. In the case of the P1 variant, the top two entries on the parameter stack are popped and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Reset

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

0: Reset

cOUT

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
(nnnn) <- P0
Z <- 1
PSC <- PSC - 1
PC <- PC + 4

Format:

nnnn cc cOUT

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the destination address.

nnnn cc cOUT

#E3

1	1	1	0	0	0	1	1

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the value in the top register of the parameter stack (P0) is popped and written to the I/O port at the address pointed to by the destination address nnnn. Execution continues at the instruction following the operand nnnn in memory.

If the condition is not met then the address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed. The contents of the parameter stack remain unchanged.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cOUTB

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
(nnnn) <- LSB P0
Z <- 1
PSC <- PSC - 1
PC <- PC + 4

Format:

nnnn cc cOUTB

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the destination address.

nnnn cc cOUTB

#E5

1	1	1	0	0	1	0	1

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the value in the top register of the parameter stack (P0) is popped and written to the I/O port at the address pointed to by the destination address nnnn. Execution continues at the instruction following the operand nnnn in memory.

If the condition is not met then the address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed. The contents of the parameter stack remain unchanged.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cpOUT

Operation: if condition cc = 0
PSC <- PSC - 2
PC <- PC + 2

if condition cc = 1
(P0) <- P1
PSC <- PSC - 2
PC <- PC + 2

Format:

cc cpOUT

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpOUT

#EB

1	1	1	0	1	0	1	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the value in the second register of the parameter stack (P1) is written to the output port at the I/O address indicated by the contents of the top register of the parameter stack (P0). If the condition is not met then the address of the next instruction in memory (PC + 2) is loaded into the PC register and points to the next program instruction to be executed. In both cases the top two entries on the parameter stack are popped and discarded.

M CYCLES: 3

Condition Bits Affected:

None

cpOUTB

Operation: if condition cc = 0
PSC <- PSC - 2
PC <- PC + 2

if condition cc = 1
(P0) <- LSB P1
PSC <- PSC - 2
PC <- PC + 2

Format:

cc cpOUTB

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpOUTB

#ED

1	1	1	0	0	1	1	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the value in the second register of the parameter stack (P1) is written to the output port at the I/O address indicated by the contents of the top register of the parameter stack (P0). If the condition is not met then the address of the next instruction in memory (PC + 2) is loaded into the PC register and points to the next program instruction to be executed. In both cases the top two entries on the parameter stack are popped and discarded.

M CYCLES: 3

Condition Bits Affected:

None

POPPS

Operation: qq <- P0
PSC <- PSC – 1
PC <- PC + 2

Format:

qq POPPS

The qq operand is any of PC, PPC, HP, FLAGS, PSP, PSC, RSP, RSC, EA or RR. These various possible opcode-operand combinations are assembled as follows in the object code:

qq POPPS

#D1

1	1	0	1	0	0	0	1

#<qq>0

<	q	q	>	0	0	0	0

Register	<qq>	
PC	0000	#00
PPC	0001	#01
HP	0010	#02
FLAGS	0011	#03
PSP	0100	#04
PSC	0101	#05
RSP	0110	#06
RSC	0111	#07
EA	1000	#08
RR	1001	#09

Description:

The contents of the top register of the parameter stack (P0) are popped into the register specified by the operand qq.

M CYCLES: 3 if PC; otherwise 2.

Condition Bits Affected:

None

POPRS

Operation: qq <- R0
RSC <- RSC – 1
PC <- PC + 2

Format:

qq POPRS

The qq operand is any of PC, PPC, HP, FLAGS, PSP, PSC, RSP, RSC, EA or RR. These various possible opcode-operand combinations are assembled as follows in the object code:

qq POPRS

#D3

1	1	1	0	0	0	1	1

#<qq>0

<	q	q	>	0	0	0	0

Register	<qq>	
PC	0000	#00
PPC	0001	#01
HP	0010	#02
FLAGS	0011	#03
PSP	0100	#04
PSC	0101	#05
RSP	0110	#06
RSC	0111	#07
EA	1000	#08
RR	1001	#09

Description:

The contents of the top register of the return stack (R0) are popped into the register specified by the operand qq.

M CYCLES: 3 if PC; otherwise 2.

Condition Bits Affected:

None

PREPARE

Operation: if condition cc = 1
HP <- (PPC)
PPC <- PPC + 2
Z <- 1
PC <- PC + 2
if condition cc = 0
Z <- 0
PC <- PC + 2

Format:

cc PREPARE

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc PREPARE

#F2

1	1	1	1	0	0	1	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then PREPARE loads the Header Pointer Register (HP) with the contents of the word in memory at the address indicated by the Pseudo Program Counter Register (PPC). The Zero Flag ("Z" in the FLAGS register) is set. The PPC register is incremented by 2. If the condition cc is not met then the Zero Flag is cleared. In both cases program execution continues at the next instruction in memory (PC + 2).

M CYCLES: 4 cycles if cc, 2 cycles if not cc

Condition Bits Affected:

Z <- 1, if condition cc is met; 0 otherwise.

PTOR

Operation: R0 <<- P0
RSC <- RSC + 1
PSC <- PSC – 1
PC <- PC + 2

Format:

PTOR

#C1

1	1	0	0	0	0	0	1

#60

0	1	1	0	0	0	0	0

Description:

The top entry of the parameter stack is popped and pushed onto the top of the return stack.

M CYCLES: 2

Condition Bits Affected:

None

PUSHPS

Operation: P0 <<- qq
PSC <- PSC + 1
PC <- PC + 2

Format:

qq PUSHPS

The qq operand is any of PC, PPC, HP, FLAGS, PSP, PSC, RSP, RSC, EA or RR. These various possible opcode-operand combinations are assembled as follows in the object code:

qq PUSHPS

#D0

1	1	0	1	0	0	0	0

#<qq>0

<	q	q	>	0	0	0	0

Register	qq	
PC	0000	#00
PPC	0001	#01
HP	0010	#02
FLAGS	0011	#03
PSP	0100	#04
PSC	0101	#05
RSP	0110	#06
RSC	0111	#07
EA	1000	#08
RR	1001	#09

Description:

The contents of the register specified by the operand qq are pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

None

PUSHRS

Operation: R0 <<- qq
RSC <- RSC + 1
PC <- PC + 2

Format:

qq PUSHRS

The qq operand is any of PC, PPC, HP, FLAGS, PSP, PSC, RSP, RSC, EA or RR. These various possible opcode-operand combinations are assembled as follows in the object code:

qq PUSHRS

#D2

1	1	0	1	0	0	1	0

#<qq>0

<	q	q	>	0	0	0	0

Register	qq	
PC	0000	#00
PPC	0001	#01
HP	0010	#02
FLAGS	0011	#03
PSP	0100	#04
PSC	0101	#05
RSP	0110	#06
RSC	0111	#07
EA	1000	#08
RR	1001	#09

Description:

The contents of the register specified by the operand qq are pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

None

RCMP

Operation: if operand is uN or sN
s – P0
PC <- PC + 2

if operand is P1
P0 – P1
PC <- PC + 2

Format:

[uN|sN] s RCMP

The s operand is any of uN , sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN RCMP

#AA

1	0	1	0	1	0	1	0

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN RCMP

#BA

1	0	1	1	1	0	1	0

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 RCMP

#C0

1	1	0	0	0	0	1	0

#B0

1	0	1	1	0	0	0	0

Description:

In the case of the sN or uN variants The contents of the top register of the parameter stack (P0) are compared to (subtracted from) the operand s, and the condition flags are set. In the case of the P1 variant the contents of the second register of the parameter stack (P1) is compared to (subtracted from) the top register of the parameter stack (P0) and the condition flags are set. The contents of the parameter stack remain unchanged.

M CYCLES: 2

Condition Bits Affected:

C: Set if there was a borrow; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

0: Set if signed overflow; reset otherwise

REFRESH

Operation: If cc = 1
(RR) <- (RR)
RR <- RR + 2
PC <- PC + 2
If cc = 0
PC <- PC + 2

Format:

cc REFRESH

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc REFRESH

#F6

1	1	1	1	0	1	1	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

This instruction is used to implement the EDAC memory wash. If the condition cc is met then a 16 bit word is read from memory at the address indicated by the contents of the Refresh Register (RR). The word is written back to memory at the same address. The RR register is then incremented by 2. If the read (or indeed any read) resulted in a bit correction by the EDAC logic, then the EDAC Error Counter (EC) in the FLAGS register will be incremented, the address of the error will be latched into the EDAC Error Address Register (EA) and the EDAC Error Flag ("EE" in the FLAGS register) will be set.

M CYCLES: 4 if cc = 1, 2 if cc = 0

Condition Bits Affected:

None

ROL

Operation: P0 <- P0, C<-b15, b15<-b14, b14<-b13 .. b0<-C
PC <- PC + 2

Format:

ROL

#C2

1	1	0	0	0	0	1	0

#20

0	0	1	0	0	0	0	0

Description:

The contents of the top register of the parameter stack (P0) are rotated left. The content of bit 0 (b0) is copied into bit 1 (b1); this pattern is continued throughout the word. The content of bit 15 (b15) is copied into the Carry Flag ("C" in the FLAGS register) and the previous content of the Carry Flag is copied into bit 0 (b0). Bit 0 (b0) is the least significant bit.

M CYCLES: 2

Condition Bits Affected:

C: Data from Bit 15 of previous contents of P0

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset.

ROR

Operation: P0 <- P0, C<-b0, b0<-b1, b1<-b2 .. b15<-C
PC <- PC + 2

Format:

ROR

#C2

1	1	0	0	0	0	1	0

#30

0	0	1	1	0	0	0	0

Description:

The contents of the top register of the parameter stack (P0) are rotated right. The content of bit 15 (b15) is copied into bit 14 (b14); this pattern is continued throughout the word. The content of bit 0 (b0) is copied into the Carry Flag ("C" in the FLAGS register) and the previous content of the Carry Flag is copied into bit 15 (b15). Bit 0 (b0) is the least significant bit.

M CYCLES: 2

Condition Bits Affected:

C: Data from Bit 0 of previous contents of P0

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset.

RSBC

Operation: if operand is uN or sN
P0 <- s – P0 – C
PC <- PC + 2

if operand is P1
P0 <- P0 – P1 – C
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s RSBC

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN RSBC

#A5

1	0	1	0	0	1	0	1

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN RSBC

#B5

1	0	1	1	0	1	0	1

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

. <			s	N			>

P1 RSBC

#C0

1	1	0	0	0	0	0	0

#30

0	0	1	1	0	0	0	0

Description:

In the case of the uN and sN variants the contents of the top register of the parameter stack (P0), along with the Carry Flag (“C” in the Flags register) are subtracted from byte or word specified by the s operand; the result replaces the contents of the P0 register. In the case of the P1 variant, the top two entries on the parameter stack are popped, the previous contents of the P1 register, along with the Carry Flag, are subtracted from the previous contents of the P0 register and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Set if there is a borrow; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

0: Set if signed overflow; reset otherwise

RSUB

Operation: if operand is uN or sN
P0 <- s – P0
PC <- PC + 2

if operand is P1
P0 <- P0 – P1
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s RSUB

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN RSUB

#A4

1	0	1	0	0	1	0	0

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN RSUB

#B4

1	0	1	1	0	1	0	0

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 RSUB

#C0

1	1	0	0	0	0	0	0

#20

0	0	1	0	0	0	0	0

Description:

In the case of the uN or sN variants the contents of the top register of the parameter stack (P0) is subtracted from the byte or word specified by the s operand; the result replaces the contents of the P0 register. In the case of the P1 variant, the top two entries on the parameter stack are popped, the previous contents of the P1 register are subtracted from the previous contents of the P0 register and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Set if there was a borrow; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

0: Set if signed overflow; reset otherwise

RTD

Operation: P0 <- P2 <- P1 <- P0
PC <- PC + 2

Format:

RTD

#C1

1	1	0	0	0	0	0	1

#50

0	1	0	1	0	0	0	0

Description:

This instruction rotates down the contents of the top three registers of the parameter stack; the old third lowest entry becomes the topmost.

M CYCLES: 2

Condition Bits Affected:

None

Example:

	P2	P1	P0
Before operation:	12	8	5
After operation:	8	5	12

RTOP

Operation: P0 <- R0
PSC <- PSC + 1
RSC <- RSC - 1
PC <- PC + 2

Format:

RTOP

#C1

1	1	0	0	0	0	0	1

#70

0	1	1	1	0	0	0	0

Description:

The top entry of the return stack is popped and pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

None

cRTS

Operation: if condition cc = 0
PC <- PC + 2

if condition cc = 1
PC <- R0
RSC <- RSC - 1

Format:

cc cRTS

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cRTS

#8E

1	0	0	0	1	1	1	0

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the contents of the top register of the return stack (R0) is popped into the Program Counter register (PC) and points to the address of the next program instruction to be executed. If the condition is not met then the address of the next instruction in memory (PC + 2) is loaded into the PC register and points to the next program instruction to be executed.

M CYCLES: 2

Condition Bits Affected:

None

RTU

Operation: P2 <- P0 <- P1 <- P2
PC <- PC + 2

Format:

RTU

#C1

1	1	0	0	0	0	0	1

#40

0	1	0	0	0	0	0	0

Description:

This instruction rotates up the contents of the top three registers of the parameter stack; the old topmost entry becomes the third lowest.

M CYCLES: 2

Condition Bits Affected:

None

Example:

	P2	P1	P0
Before operation:	8	5	12
After operation:	12	8	5

SBC

Operation: if operand is uN or sN
P0 <- P0 - s – C
PC <- PC + 2

if operand is P0
P0 <- P1 – P0 – C
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s SBC

The s operand is any of uN, sN or P0. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN SBC

#A3

1	0	1	0	0	0	1	1

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN SBC

#B3

1	0	1	1	0	0	1	1

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P0 SBC

#C0

1	1	0	0	0	0	0	0

#50

0	1	0	1	0	0	0	0

Description:

In the case of the uN or sN variants the word specified by the s operand, along with the Carry Flag (“C” in the Flags register) is subtracted from the contents of the top register of the parameter stack (P0); the result replaces contents of the P0 register. In the case of the P0 variant, the top two entries on the parameter stack are popped, the previous contents of the P0 register and the Carry Flag are subtracted from the P1 register and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Set if there was a borrow; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

0: Set if signed overflow; reset otherwise

SET

Operation: FLAGS <<- FLAGS OR m
PC <- PC + 2

Format:

m SET

#D4

1	1	0	1	0	1	0	0

#<m>0

<	m		>	0	0	0	0

Description:

The bit corresponding to the value of m in the FLAGS register is set according to the following table:

FLAGS bit		m	
C		0000	#00
Z		0001	#01
S		0010	#02
O		0011	#03
E		0100	#04
I	*	0101	#05
IE		0110	#06
EE	*	0111	#07

Setting of bits marked with an asterisk may produce unpredictable results.

M CYCLES: 2

Condition Bits Affected:

The bit indicated by the m operand is set. Setting the E, EE, or I flags is not recommended.

SOT

Operation: P0 <<- P1
PSC <- PSC + 1
PC <- PC + 2

Format:

SOT

#C1

1	1	0	0	0	0	0	1

#30

0	0	1	1	0	0	0	0

Description:

The contents of the second register of the parameter stack (P1) are pushed onto the top of the parameter stack; i.e. duplicates the second entry on the top of the stack.

M CYCLES: 2

Condition Bits Affected:

None

Example:

	P2	P1	P0
Before operation:	-	5	12
After operation:	5	12	5

cSTORE

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
(nnnn) <- P0
Z <- 1
PSC <- PSC - 1
PC <- PC + 4

Format:

nnnn cc cSTORE

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the destination address.

nnnn cc cSTORE

#83

1	0	0	0	0	0	1	1

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the value in the top register of the parameter stack (P0) is popped and stored in memory at the address pointed to by the destination address nnnn. Execution continues at the instruction following the operand nnnn in memory.

If the condition is not met then the address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed. The contents of the parameter stack remain unchanged.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cSTOREB

Operation: if condition cc = 0
Z <- 0
PC <- PC + 4

if condition cc = 1
(nnnn) <- LSB P0
Z <- 1
PSC <- PSC - 1
PC <- PC + 4

Format:

nnnn cc cSTOREB

The cc operand is any of the condition codes as defined for the FLAG instruction. nnnn is the destination address.

nnnn cc cSTOREB

#85

1	0	0	0	0	1	0	1

#0<cc>

0	0	0	0	<	c	c	>

LSB nnnn

<			n	n			>

MSB nnnn

<			n	n			>

Description:

If the condition cc is met then the top entry of the parameter stack (P0) is popped, and the least significant byte is stored in memory at the address pointed to by the destination

address nnnn. Execution continues at the instruction following the operand nnnn in memory.

If the condition is not met then the address of the instruction following the operand nnnn in memory (PC + 4) is loaded into the PC register and points to the next program instruction to be executed. The contents of the parameter stack remain unchanged.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cpSTORE

Operation: if condition cc = 0
Z <- 0
PSC <- PSC - 2
PC <- PC + 2

if condition cc = 1
(P0) <- P1
Z <- 1
PSC = PSC - 2
PC <- PC + 2

Format:

cc cpSTORE

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpSTORE

#8B

1	0	0	0	1	0	1	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the contents of the second register of the parameter stack (P1) are stored in memory at the address pointed to by the top register of the parameter stack (P0). In both cases the top two entries in the parameter stack are popped.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

cpSTOREB

Operation: if condition cc = 0
Z <- 0
PSC <- PSC - 2
PC <- PC + 2

if condition cc = 1
(P0) <- LSB P1
Z <- 1
PSC = PSC - 2
PC <- PC + 2

Format:

cc cpSTOREB

The cc operand is any of the condition codes as defined for the FLAG instruction.

cc cpSTOREB

#8D

1	0	0	0	1	1	0	1

#0<cc>

0	0	0	0	<	c	c	>

Description:

If the condition cc is met then the contents of the least significant byte of the second register of the parameter stack (P1) are stored in memory at the address pointed to by the top register of the parameter stack (P0). In both cases the top two entries of the parameter stack are popped.

M CYCLES: 3

Condition Bits Affected:

Z Set if condition met, reset otherwise.

sSTORE

Operation: (abc) <- P0
PC <- PC + 2

Format:

abc sSTORE

#3<a>

0	0	1	1	<	a		>

<b><c>

<	b		>	<	c		>

Description:

The contents of the top register of the parameter stack (P0) are stored in the address in memory location indicated by the operand abc. The contents of the parameter stack remain unchanged.

M CYCLES: 3

Condition Bits Affected:

None

SUB

Operation: if operand is uN or sN
P0 <- P0 – s
PC <- PC + 2

if operand is P0
P0 <- P1 – P0
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s SUB

The s operand is any of uN, sN or P0. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN SUB

#A2

1	0	1	0	0	0	1	0

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN SUB

#B2

1	0	1	1	0	0	1	0

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P0 SUB

#C0

1	1	0	0	0	0	0	0

#40

0	1	0	0	0	0	0	0

Description:

In the case of the uN or sN variants the word specified by the s operand is subtracted from the top of the parameter stack (P0); the result replaces the contents of the P0 register. In the case of the P0 variant, the top two entries on the parameter stack are popped, the previous contents of the P0 register are subtracted from the previous contents of the P1 register and the result pushed onto the top of the parameter stack.

M CYCLES: 2

Condition Bits Affected:

C: Set if there was a borrow; reset otherwise

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

0: Set if signed overflow; reset otherwise

SWAP

Operation: P0 <-> P1
PC <- PC + 2

Format:

SWAP

#C1

1	1	0	0	0	0	0	1

#20

0	0	1	0	0	0	0	0

Description:

The contents of the top two registers of the parameter stack (P0 and P1) are exchanged; i.e. swapped.

M CYCLES: 2

Condition Bits Affected:

None

TST

Operation: if operand is uN or sN
P0 AND s
PC <- PC + 2

if operand is P1
P0 AND P1
PSC <- PSC - 1
PC <- PC + 2

Format:

[uN|sN] s TST

The s operand is any of uN, sN or P1. These various possible opcode-operand combinations are assembled as follows in the object code:

uN uN TST

#AE

1	0	1	0	1	1	1	0

8-bit unsigned value, this is extended by the processor prior to the operation by filling bits 8 – 15 with zeroes.

<			u	N			>

sN sN TST

#BE

1	0	1	1	1	1	1	0

8-bit signed value, this is extended by the processor prior to the operation by filling bits 8 – 15 with the most significant bit of the operand.

<			s	N			>

P1 TST

#C0

1	1	0	0	0	0	0	0

#E0

1	1	1	0	0	0	0	0

Description:

A logical AND operation is performed, bit by bit, between the byte or word specified by the s operand and the contents of the top register of the parameter stack (P0); the condition flags are set. The contents of the stack remain unchanged.

M CYCLES: 2

Condition Bits Affected:

C: Reset

S: Set if result is negative; reset otherwise

Z: Set if result is zero; reset otherwise

O: Reset.

XB

Operation: P0 <- P0<LSB>P0<MSB>
PC <- PC + 2

Format:

XB

#FD

1	1	1	1	1	1	0	1

#00

0	0	0	0	0	0	0	0

Description:

The XB instruction exchanges the order of the two bytes in the top register of the parameter stack (P0). i.e. the LSB becomes the MSB and vice versa.

M CYCLES: 2

Condition Bits Affected:

None

XRP

Operation: R0 <-> P0
PC <- PC + 2

Format:

XRP

#C1

1	1	0	0	0	0	0	1

#90

1	0	0	1	0	0	0	0

Description:

The contents of the top registers of the parameter stack (P0) and the return stack (R0) are exchanged; i.e. swapped, there is no pushing or popping.

M CYCLES: 2

Condition Bits Affected:

None

CHAPTER 4 - Am1601 Assembler / IPS-F1G

The IPS-F1G source code must be compiled with a version of IPS-X that supports the “align” variable.

```
(                       Am1601 Assembler                        )
(                    Copyright 2002 AMSAT-DL                    )
(                 by Karl Meinzer, James Miller,                )
(                  Lyle Johnson & Paul Willmott                 )

(    This program is free software; you can redistribute it     )
(    and/or modify it under the terms of the GNU General        )
(    Public License as published by the Free Software           )
(    Foundation; either version 2 of the License, or at         )
(    your option, any later version.                            )

(    This program is distributed in the hope that it will       )
(    be useful, but WITHOUT ANY WARRANTY; without even the      )
(    implied warranty of MERCHANTABILITY or FITNESS FOR A       )
(    PARTICULAR PURPOSE.  See the GNU General Public            )
(    License for more details.                                  )

(    You should have received a copy of the GNU General         )
(    Public License along with this program; if not, write      )
(    to the Free Software Foundation, Inc., 59 Temple           )
(    Place, Suite 330, Boston, MA  02111-1307  USA              )

(                Contact : vp9mu@amsat.org                      )


(    NOTE: stack comments have top on right                     )

(   "Assembler" definitions for use by IPS-X cross compiler    )

:prior i> 0 compileflag !b ;n
:int <i 1 compileflag !b ;n
:n , hier $OC !b $h incr ;n
:int code  entrysetup ja? hier vert !O
                      dann ;n
:int rcode entrysetup ja? !O dann ;n

02 align !n                       ( set even address alignment )

(                  Constants for cc codes                       )
(                  ----------------------                       )

#0 kon EQ #1 kon NE #2 kon CS #3 kon CC
#4 kon MI #5 kon PL #6 kon VS #7 kon VC
#8 kon HS #9 kon LO #A kon GE #B kon LT
#C kon GT #D kon LE #E kon AL #F kon NEF
#2 kon HI #3 kon LS
```

```
( Comparison  Unsigned   Signed )
( =           EQ         EQ     )
( !=          NE         NE     )
( >=          HS         GE     )
( >           HI CS      GT     )
( <=          LS CC      LE     )
( <           LO         LT     )


(                Constants for PUSH & POP                   )
(                ------------------------                   )

#00 kon PC  #10 kon PPC #20 kon HP  #30 kon FLAGS
#40 kon PSP #50 kon PSC #60 kon RSP #70 kon RSC
#80 kon EA  #90 kon RR

(     Constants for Arithmetic/Logical Instruction Operands   )
(     -----------------------------------------------------   )

#00A0 kon uN                          ( unsigned operand flag )
#00B0 kon sN                            ( signed operand flag )
#00C0 kon P0                ( parameter stack register 0 flag )
#00C0 kon P1                ( parameter stack register 1 flag )

(          Constants for SET & CLEAR Instructions             )
(          --------------------------------------             )

#00 kon FLGC #10 kon FLGZ #20 kon FLGS  #30 kon FLGO
#40 kon FLGE #50 kon FLGI #60 kon FLGIE #70 kon FLGEE

(           byte manipulation and storage primitives          )
(           ----------------------------------------          )

:n sJCODE    ( <addr> <opcode>           )
   vert      ( <opcode> <addr>           )
   dup       ( <opcode> <addr> <addr>    )
   #100 /n   ( <opcode> <addr> <MSBAddr> )
   #0F und   ( restrict range to 0-F     )
   rdo       ( <addr> <MSBaddr> <opcode> )
   oder ,    ( <LSBaddr>                 )
   #FF und , ( -                         )
;n

:n ccCODE , #0F und , ;n

:n aluCODE        ( <P1/0> <subc>             )
      zwo         ( <P1/0> <subc> <P1/0>      )
      P1 =n ja?   ( <P1/0> <subc>             )
         256 *n   ( <P1/0> <subc>*256         )
         oder     ( <opcode>                  )
         $dep     ( -                         )
```

```
      nein:       ( <numb> <uN|sN> <subc>     )
         16 /n    ( <numb> <uN|sN> <subc/16>  )
         oder     ( <numb> <opcode>           )
         ,        ( <numb>                    )
         ,        ( -                         )
      dann
;n

(                     instructions                          )
(                     ------------                          )

:n sJMP   #00 sJCODE ;n
:n sJSR   #10 sJCODE ;n
:n sLOAD  #20 sJCODE ;n
:n sSTORE #30 sJCODE ;n
:n sBR    #40 sJCODE ;n
:n sBSR   #50 sJCODE ;n
:n cNLOAD #60 ccCODE $dep ;n
:n uNLOAD #70 , , ;n
:n sNLOAD #71 , , ;n

:n cJSR     #80 ccCODE $dep ;n
:n cJMP     #81 ccCODE $dep ;n
:n cLOAD    #82 ccCODE $dep ;n
:n cSTORE   #83 ccCODE $dep ;n
:n cLOADB   #84 ccCODE $dep ;n
:n cSTOREB  #85 ccCODE $dep ;n
:n cpJSR    #88 ccCODE ;n
:n cpJMP    #89 ccCODE ;n
:n cpLOAD   #8A ccCODE ;n
:n cpSTORE  #8B ccCODE ;n
:n cpLOADB  #8C ccCODE ;n
:n cpSTOREB #8D ccCODE ;n
:n cRTS     #8E ccCODE ;n
:n cpBSR    #8F ccCODE ;n

:n cBR #90 oder , , ;n

:n ADD  #00 aluCODE ;n
:n ADC  #10 aluCODE ;n

:n SBC  dup P1 =n ja?
            #50
        nein:
            #30
        dann
        aluCODE ;n


:n SUB  dup P1 =n ja?
            #40
        nein:
```

```
            #20
        dann
        aluCODE ;n

:n RSBC dup P1 =n ja?
            #30
        nein:
            #50
        dann
        aluCODE ;n

:n RSUB dup P1 =n ja?
            #20
        nein:
            #40
        dann
        aluCODE ;n

:n AND  #60 aluCODE ;n
:n OR   #70 aluCODE ;n
:n EOR  #80 aluCODE ;n
:n NOP  #90C0 $dep ;n

:n CMP  dup P1 =n ja?
            #A0
        nein:
            #B0
        dann
        aluCODE ;n

:n RCMP dup P1 =n ja?
            #B0
        nein:
            #A0
        dann
        aluCODE ;n

:n MASK #C0 aluCODE ;n
:n CPL  #D0C0 $dep ;n
:n TST  #E0 aluCODE ;n
:n NEG  #F0C0 $dep ;n

:n DUPL #00C1 $dep ;n
:n DEL  #10C1 $dep ;n
:n SWAP #20C1 $dep ;n
:n SOT  #30C1 $dep ;n
:n RTU  #40C1 $dep ;n
:n RTD  #50C1 $dep ;n
:n PTOR #60C1 $dep ;n
:n RTOP #70C1 $dep ;n
:n IDX  #80C1 $dep ;n
:n XRP  #90C1 $dep ;n
```

```
:n LSL  #00C2 $dep ;n
:n LSR  #10C2 $dep ;n
:n ROL  #20C2 $dep ;n
:n ROR  #30C2 $dep ;n
:n ASR  #90C2 $dep ;n

:n PUSHPS #D0 , , ;n
:n POPPS  #D1 , , ;n
:n PUSHRS #D2 , , ;n
:n POPRS  #D3 , , ;n
:n SET    #D4 , , ;n
:n CLEAR  #D5 , , ;n

:n cIN    #E2 ccCODE $dep ;n
:n cOUT   #E3 ccCODE $dep ;n
:n cINB   #E4 ccCODE $dep ;n
:n cOUTB  #E5 ccCODE $dep ;n
:n cpIN   #EA ccCODE ;n
:n cpOUT  #EB ccCODE ;n
:n cpINB  #EC ccCODE ;n
:n cpOUTB #ED ccCODE ;n

:n EMULATE #F0 ccCODE ;n
:n EXECUTE #F1 ccCODE ;n
:n PREPARE #F2 ccCODE ;n
:n REFRESH #F6 ccCODE ;n
:n DFX     #00F8 $dep ;n
:n 2BLIT   #00FB $dep ;n
:n JPPC    #00FC $dep ;n
:n XB      #00FD $dep ;n
:n FLAG    #FF ccCODE ;n

(                 Jump and Branch Tools                    )
(                 ---------------------                    )

:n sJSRbegin
     hier          ( push address onto IPS-X stack   )
     h2inc         ( deposit placeholder             )
;n                 ( <fixaddr>                       )

:n sJSRcomplete    ( <fixaddr>                       )
     hier          ( <fixaddr> <saveaddr>            )
     dup           ( <fixaddr> <saveaddr> <saveaddr> )
     rdo           ( <saveaddr> <saveaddr> <fixaddr> )
     $h !n         ( <saveaddr> <saveaddr>           )
     sJSR          ( <saveaddr>                      )
     $h !n         ( -                               )
;n

:n sBRbegin
     hier          ( push address onto IPS-X stack   )
     h2inc         ( deposit placeholder             )
```

```
;n                ( <fixaddr>                        )

:n sBRcomplete    ( <fixaddr>                        )
     dup          ( <fixaddr> <fixaddr>              )
     02 +n        ( <fixaddr> <PC+2>                 )
     hier         ( <fixaddr> <PC+2> <jumpadd>       )
     vert         ( <fixaddr> <jumpadd> <PC+2>       )
     -n           ( <fixaddr> <offset>               )
     hier         ( <fixaddr> <offset> <saveaddr>    )
     rdu          ( <saveaddr> <fixaddr> <offset>    )
     vert         ( <saveaddr> <offset> <fixaddr>    )
     $h !n        ( <saveaddr> <offset>              )
     sBR          ( <saveaddr>                       )
     $h !n        ( -                                )
;n

:n sJMPbegin
     hier         ( push address onto IPS-X stack    )
     h2inc        ( deposit placeholder              )
;n                ( <fixaddr>                        )

:n sJMPcomplete   ( <fixaddr>                        )
     hier         ( <fixaddr> <saveaddr>             )
     dup          ( <fixaddr> <saveaddr> <saveaddr>  )
     rdo          ( <saveaddr> <saveaddr> <fixaddr>  )
     $h !n        ( <saveaddr> <saveaddr>            )
     sJMP         ( <saveaddr>                       )
     $h !n        ( -                                )

;n

:n sBSRbegin
     hier         ( push address onto IPS-X stack    )
     h2inc        ( deposit placeholder              )
;n                ( <fixaddr>                        )

:n sBSRcomplete   ( <fixaddr>                        )
     dup          ( <fixaddr> <fixaddr>              )
     02 +n        ( <fixaddr> <PC+2>                 )
     hier         ( <fixaddr> <PC+2> <jumpadd>       )
     vert         ( <fixaddr> <jumpadd> <PC+2>       )
     -n           ( <fixaddr> <offset>               )
     hier         ( <fixaddr> <offset> <saveaddr>    )
     rdu          ( <saveaddr> <fixaddr> <offset>    )
     vert         ( <saveaddr> <offset> <fixaddr>    )
     $h !n        ( <saveaddr> <offset>              )
     sBSR         ( <saveaddr>                       )
     $h !n        ( -                                )
;n

:n cJMPbegin      ( <cc>                             )
     #81 , ,      ( - ) ( cJMP opcode deposited      )
```

```
     hier          ( push address onto IPS-X stack   )
     h2inc         ( leave placeholder for address   )
;n                 ( <fixaddr>                       )

:n cJMPend         ( <fixaddr>                       )
     hier          ( <fixaddr> <saveaddr>            )
     vert          ( <saveaddr> <fixaddr>            )
     $OC !n        ( -                               )
;n

:n cJMPelse        ( <fixaddr>                       )
     AL cJMPbegin  ( <fixaddr> <fixaddr2>            )
     vert          ( <fixaddr2> <fixaddr>            )
     cJMPend       ( <fixaddr2>                      )
;n

:n cJSRbegin       ( <cc>                            )
     #80 , ,       ( - ) ( cJSR opcode deposited     )
     hier          ( push address onto IPS-X stack   )
     h2inc         ( leave placeholder for address   )
;n                 ( <fixaddr>                       )

:n cJSRcomplete    ( <fixaddr>                       )
     hier          ( <fixaddr> <saveaddr>            )
     vert          ( <saveaddr> <fixaddr>             )
     $OC !n        ( -                               )
;n


:n cBRbegin        ( <cc>                            )
     #90 oder ,    ( - ) ( cc cBR opcode deposited   )
     hier          ( push address onto IPS-X stack   )
     #0 ,          ( leave placeholder for offset    )
;n                 ( <fixaddr>                       )

:n cBRend          ( <fixaddr>                       )
     dup           ( <fixaddr> <fixaddr>             )
     01 +n         ( <fixaddr> <PC+2>                )
     hier          ( <fixaddr> <PC+2> <jumpadd>      )
     vert          ( <fixaddr> <jumpadd> <PC+2>      )
     -n            ( <fixaddr> <offset>              )
     vert          ( <offset> <fixaddr>              )
     $OC !b        ( -                               )
;n

:n cBRelse         ( <fixaddr>                       )
     AL cBRbegin   ( <fixaddr> <fixaddr2>            )
     vert          ( <fixaddr2> <fixaddr>            )
     cBRend        ( <fixaddr2>                      )
;n

( these are the traditional IPS Assembler Definitions )
```

```
:n Y? cJMPbegin ;n
:n N: cJMPelse ;n
:n TH cJMPend ;n
:n BEGIN hier ;n
:n END Y? $OC !n ;n
:n TH/AGAIN vert AL END TH ;n

(                   End Am1601 Assembler                    )
(                   --------------------                    )



(                     IPS-F1G for Am1601                    )
(                   Copyright 2002 AMSAT-DL                 )
(                by Karl Meinzer, James Miller,             )
(                 Lyle Johnson & Paul Willmott              )

(    This program is free software; you can redistribute it  )
(    and/or modify it under the terms of the GNU General     )
(    Public License as published by the Free Software        )
(    Foundation; either version 2 of the License, or at      )
(    your option, any later version.                         )

(    This program is distributed in the hope that it will    )
(    be useful, but WITHOUT ANY WARRANTY; without even the   )
(    implied warranty of MERCHANTABILITY or FITNESS FOR A    )
(    PARTICULAR PURPOSE.  See the GNU General Public         )
(    License for more details.                               )

(    You should have received a copy of the GNU General      )
(    Public License along with this program; if not, write   )
(    to the Free Software Foundation, Inc., 59 Temple        )
(    Place, Suite 330, Boston, MA  02111-1307  USA           )

(                Contact : vp9mu@amsat.org                  )

(    NOTE: Lyle's stack comments have top on left            )
(          Paul's stack comments have top on right           )

~      Compiling IPS-F1G      ~ #01D5 !t ( Information splash )


(                     IPS-F1G Memory Map                    )
(                     ------------------                    )

( #0000 Reset Vector                                        )
( #0004 Return Stack Underflow Vector                       )
( #0008 Parameter Stack Underflow Vector                    )
( #000C PC Odd Vector                                       )
( #0010 Maskable Interrupt Vector                           )
```

```
( #0100 Screen                                                  )
( #0500 Syspage                                                 )

(                Syspage assignments 500-57F                    )
(                --------------------------                     )

( 500  ' COMPILER                    520  SU0 Minutes LSW       )
( 501                                521  SU0 Minutes MSW       )
( 502  0 ' ZEIG-STAPEL <KETTE>       522  SU1 10ms    <0-98>    )
( 503                                523  SU1 Seconds <0-59>    )
( 504  1                            524  SU1 Minutes LSW       )
( 505                                525  SU1 Minutes MSW       )
( 506  2                            526  SU2 10ms    <0-98>    )
( 507                                527  SU2 Seconds <0-59>    )
( 508  3                            528  SU2 Minutes LSW       )
( 509                                529  SU2 Minutes MSW       )
( 50A  4                            52A  SU3 10ms    <0-98>    )
( 50B                                52B  SU3 Seconds <0-59>    )
( 50C  5                            52C  SU3 Minutes LSW       )
( 50D                                52D  SU3 Minutes MSW       )
( 50E  6                            52E  READYFLAG             )
( 50F                                52F  LOADFLAG              )
( 510  7                            530  $PE value             )
( 511                                531                       )
( 512  JUMP                          532  $PI value             )
( 513                                533                       )
( 514  #0500                         534  $P1 value             )
( 515                                535                       )
( 516  --                            536  $P2 value             )
( 517  --                            537                       )
( 518  UHR 10ms    <0-98>           538  $P3 value             )
( 519  UHR Seconds <0-59>           539                       )
( 51A  UHR Minutes <0-59>           53A  $H  value             )
( 51B  UHR Hours   <0-23>           53B                       )
( 51C  UHR Days LSW                 53C  $Os value            )
( 51D  UHR Days MSW                 53D                       )
( 51E  SU0 10ms    <0-98>           53E  $ND value            )
( 51F  SU0 Seconds <0-59>           53F                       )

( #0540 - #054F reserved for 20ms use                           )

( 540  Keyboard Input Pointer                                   )
( 541                                                           )
( 542  Insert Flag                                              )

( #0600 Reset Service Routine                                   )

( #0650 20ms Service Routine                                    )

( #FEFE Parameter Stack Overflow Start                          )
( #FFFE Return Stack Overflow Start                             )
```

```
( Constants in the IPS-X definition list for the compilation  )
(                of the code and 20ms routines                )


#0100 kon $$tv0               ( 1st   TV screen line position )
#0200 kon $$tv4               ( 4th   TV screen line position )
#0300 kon $$tv8               ( 8th   TV screen line position )
#0100 kon $$tvs               ( Stack TV screen line position )
#04FF kon $$tve               ( Last  TV screen line position )
#052E kon $$readyflag                  ( Compiler READYFLAG )
#0530 kon $$pe                      ( $PE address in syspage )
#0532 kon $$pi                      ( $PI address in syspage )
#0540 kon $$kbdip                   ( Keyboard Input Pointer )
#0542 kon $$inson                     ( Insert Mode On Flag )
#0    kon $$kpport        ( Keyboard Pressed I/O Port Address )
#2    kon $$kvport       ( Key Pressed Value I/O Port Address )
#0650 kon $$20ms                     ( 20ms Service Routine )
:n NEXT NEF EMULATE $$20ms sJMP ;n

X>>                                    ( Enter compile mode )

#0000 $h !n
#0 hier !O hier $OC dup 1 +n #3FFF l>>>         ( Wipe memory )

#0000 $h !n #0600 sJMP                        ( Reset Vector )
#0004 $h !n
#0008 $h !n
#000C $h !n
#0010 $h !n

(               Fill screen buffer with spaces               )
(               -----------------------------                )

#20 $$tv0 !O $$tv0 $OC dup 1 +n #3FF l>>>


~   IPS-F1G 2002-Oct-28c   ~ #02D5 $OC !t        ( Identifier )

#0518 $h !n                                            ( UHR )
0 , 0 , 0 , 0 , 0 , 0 ,

#051E $h !n                                    ( Stop watches )
1 , 0 , 0 , 0 ,  1 , 0 , 0 , 0 ,
1 , 0 , 0 , 0 ,  1 , 0 , 0 , 0 ,

(             #540-#54F free for implementor's use           )
(             ------------------------------------           )

#0600 $h !n                          ( Reset Service Routine )
#FFFE AL cNLOAD RSP POPPS                          ( Set RSP )
#0004 AL cNLOAD RSC POPPS            ( Max 4 items underflow )
#FEFE AL cNLOAD PSP POPPS                          ( Set PSP )
```

```
#0004 AL cNLOAD PSC POPPS             ( Max 4 items underflow )
#0500 AL cNLOAD PPC POPPS                              ( Set PPC )
AL EMULATE                            ( Start it Running !!! )

$$20ms $h !n

( UHR & Stopwatches )

sJSRbegin ( UHR )
#051E AL cNLOAD sJSRbegin ( SU0 StopWatch )
#0522 AL cNLOAD sJSRbegin ( SU1 StopWatch )
#0526 AL cNLOAD sJSRbegin ( SU2 StopWatch )
#052A AL cNLOAD sJSRbegin ( SU3 StopWatch )

$$readyflag AL cLOADB
#01 uN AND DEL EQ cJSRbegin ( Keyboard Handler )

AL REFRESH
FLGE CLEAR
AL EMULATE

(   The Keyboard Handler processes one key press before )
(   returning control to 20ms routine.                  )

cJSRcomplete ( Keyboard Handler )

    $$kpport AL cINB      ( KyP              )
    #1 uN AND             ( KyP              )
    DEL                   (                  )
    EQ cRTS               ( Quit if no key   )

(   The previous blob cursor is removed. The Blob      )
(   cursor uses the fact that setting the MS bit of a  )
(   character in the iPS screen area causes it to be   )
(   displayed in reverse video.                        )

    $$kbdip AL cLOAD      ( Adr              )
    DUPL                  ( Adr Adr          )
    AL cpLOADB            ( Adr Chr          )
    #7F uN AND            ( Adr Chr          )
    SWAP                  ( Chr Adr          )
    AL cpSTOREB           ( -                )

(   Get key value from Input Port )

    $$kvport AL cIN       ( Chr              )
    #00 uNLOAD            ( 0 Chr            )
    $$kpport AL cOUTB     ( Chr              )

(   PC Control Keys: If the key pressed is not an    )
(   ASCII key, e.g. the cursor keys. Then the        )
(   hardware returns #FF in the MSB of the keyboard  )
```

```
(   buffer, otherwise #00.                          )

    DUPL                ( Chr Chr          )
    #FF uNLOAD          ( Chr Chr #FF      )
    XB                  ( Chr Chr #FF00    )
    P1 AND DEL          ( Chr              )
    #02 NE cBR          ( Chr              )

( branch to nCtrl: )
    sBRbegin            ( Chr              )

    #FF uN AND          ( Chr              )

( Chk71: Home                                       )
( Move the blob cursor to the start of the IPS input )
( screen area                                       )
( ------------------------------------------------- )

    71 uN CMP           ( Chr              )

( branch to Chk72: )
    12 NE cBR           ( Chr              )
       DEL              (                  )
       $$tv8 AL cNLOAD  ( TV8              )
       $$kbdip AL cSTORE (                 )
( branch to PtrLim: )
       sBRbegin

( Chk72: Up Arrow                 )
( Move the blob cursor up one line )
( ------------------------------- )

    72 uN CMP           ( Chr              )
( branch to Chk75: )
    14 NE cBR           ( Chr              )

       DEL              (                  )
       $$kbdip AL cLOAD ( KIP              )
       64 uN SUB        ( KIP              )
       $$kbdip AL cSTORE (                 )
( branch to PtrLim: )
       sBRbegin

( Chk75: Left Arrow                                 )
( Move the blob cursor 1 character position to the  )
( left.                                             )
( ------------------------------------------------- )

    75 uN CMP           ( Chr          )
( branch to Chk75: )
    14 NE cBR           ( Chr          )
```

```
       DEL                (               )
       $$kbdip AL cLOAD   ( KIP           )
       01 uN SUB          ( KIP           )
       $$kbdip AL cSTORE  (               )
( branch to PtrLim: )
       sBRbegin

( Chk77: - Right Arrow  )
( Move the blob cursor right 1 character position  )
( ------------------------------------------------  )

    77 uN CMP             ( Chr              )
( branch to Chk79: )
    14 NE cBR             ( Chr              )

       DEL                (                  )
       $$kbdip AL cLOAD   ( KIP              )
       01 uN ADD          ( KIP              )
       $$kbdip AL cSTORE  (                  )
( branch to PtrLim: )
       sBRbegin

( Chk79: End )
( Move the blob cursor to the end of the IPS Input )
( Area.                                            )
( ------------------------------------------------ )

    79 uN CMP             ( Chr              )
( branch to Chk80: )
    12 NE cBR             ( Chr              )

       DEL                (                  )
       $$tve AL cNLOAD    ( TVE              )
       $$kbdip AL cSTORE  (                  )
( branch to PtrLim: )
       sBRbegin

( Chk80: - Down Arrow                 )
( Move the blob cursor down one line. )
( ---------------------------------- )

    80 uN CMP             ( Chr              )
( branch to Chk82: )
    14 NE cBR             ( Chr              )

       DEL                (                  )
       $$kbdip AL cLOAD   ( KIP              )
       64 uN ADD          ( KIP              )
       $$kbdip AL cSTORE  (                  )
( branch to PtrLim: )
       sBRbegin
```

```
( Chk82: - Insert              )
( Toggle the insert mode flag. )
( ---------------------------- )


    82 uN CMP             ( Chr             )
( branch to Chk83: )
    14 NE cBR             ( Chr             )

       DEL                (                 )
       $$inson AL cLOADB  ( Ins             )
       #01 uN EOR         ( ~Ins            )
       $$inson AL cSTOREB (                 )
( branch to PtrLim: )
       sBRbegin

( Chk83: - Delete                             )
( Delete the character under the blob cursor. )
( ------------------------------------------- )

    83 uN CMP             ( Chr             )
( branch to Chk45: )
    48 NE cBR             ( Chr             )

       DEL                (                 )
       $$tve AL cNLOAD    ( TVE             )
       $$kbdip AL cLOAD   ( TVE KIP         )
       P0 CMP             ( TVE KIP         )
( branch to PutSpc: )
       EQ cBRbegin        ( TVE KIP         )

( Loop: )
       DUPL               ( P: KIP KIP TVE      R: )
       01 uN ADD          ( P: KI+ KIP TVE      R: )
       DUPL               ( P: KI+ KI+ KIP TVE  R: )
       PTOR               ( P: KI+ KIP TVE      R: KI+ )
       AL cpLOADB         ( P: Suc KIP TVE      R: )
       SWAP               ( P: KIP Suc TVE      R: KI+ )
       AL cpSTOREB        ( P: TVE              R: KI+ )
       DUPL               ( P: TVE TVE          R: KI+ )
       RTOP               ( P: KI+ TVE TVE      R: )
       DUPL               ( P: KI+ KI+ TVE TVE  R: )
       RTD                ( P: TVE KI+ KI+ TVE  R: )
       P0 SUB             ( P: Dif KI+ TVE      R: )
       DEL                ( P: KI+ TVE          R: )
( branch to Loop: )
       -28 NE cBR         ( P: KI+ TVE          R: )

( PutSpc: )
       cBRend
       DEL                ( TVE             )
       32 uNLOAD          ( TVE 32          )
```

```
       SWAP               ( 32 TVE           )
       AL cpSTOREB        (                  )
( branch to PtrLim: )

( PtrLim: INPUTPOINTER := INPUTPOINTER AND $3FF )
( offset by +#0100 for Am1601 ...               )
( ---------------------------------------------- )

sBRcomplete ( Insert )
sBRcomplete ( Down Arrow )
sBRcomplete ( End )
sBRcomplete ( Right Arrow )
sBRcomplete ( Left Arrow )
sBRcomplete ( Up Arrow )
sBRcomplete ( Home )

    $$kbdip AL cLOAD      ( P: KIP             R: )
    #100 AL cNLOAD        ( P: 100 KIP         R: )
    DUPL                  ( P: 100 100 KIP     R: )
    RTD                   ( P: KIP 100 100     R: )
    P1 RSUB               ( P: KIP 100         R: )
    #3FF AL cNLOAD        ( P: 3FF KIP 100     R: )
    P1 AND                ( P: KIP 100         R: )
    P1 ADD                ( P: KIP             R: )
    $$kbdip AL cSTORE     ( P:                 R: )
( branch to PutBlob: )
    220 sBR               (                  )

( nCtrl: ASCII Characters )
sBRcomplete

( CkkCR: Carriage Return / Enter - Start Compiler )
( ---------------------------------------------- )

    13 uN CMP             ( Chr              )
( branch to ChkBS: )
    14 NE cBR             ( Chr              )

    DEL                   (                  )
    $$kbdip AL cLOAD      ( KIP              )
    01 uN SUB             ( KI-              )
    $$pe AL cSTORE        (                  )
( branch to PutBlob: )
    202 sBR               (                  )

( ChkBS: BackSpace                                  )
( Delete the character to the left of the blob cursor )
( ------------------------------------------------- )

    8 uN CMP              ( Chr              )
( branch to Other: )
    94 NE cBR             ( Chr              )
```

```
    DEL                 (                   )
    $$kbdip AL cLOAD    ( P: KIP              R: )
    01 uN SUB           ( P: KIP              R: )
    #100 AL cNLOAD      ( P: 100 KIP          R: )
    DUPL                ( P: 100 100 KIP      R: )
    RTD                 ( P: KIP 100 100      R: )
    P1 RSUB             ( P: KIP 100          R: )
    #3FF AL cNLOAD      ( P: 3FF KIP 100      R: )
    P1 AND              ( P: KIP 100          R: )
    P1 ADD              ( P: KIP              R: )
    DUPL                ( P: KIP KIP          R: )
    $$kbdip AL cSTORE   ( P: KIP              R: )

    32 uNLOAD           ( P: 32 KIP           R: )
    SWAP                ( P: KIP 32           R: )
    AL cpSTOREB         ( P:                  R: )


(   IF InsertON THEN )
    $$inson AL cLOADB   ( P: Ins              R: )
    01 uN AND           ( P: Ins              R: )
    DEL                 ( P:                  R: )
( branch to Other: )
    44 EQ cBR           ( P:                  R: )

(   FOR Index := INPUTPOINTER TO TVE DO )
    $$tve AL cNLOAD     ( P: TVE              R: )
    $$kbdip AL cLOAD    ( P: KIP TVE          R: )

( Loop: )
       DUPL             ( P: KIP KIP TVE      R: )
       01 uN ADD        ( P: KI+ KIP TVE      R: )
       DUPL             ( P: KI+ KI+ KIP TVE  R: )
       PTOR             ( P: KI+ KIP TVE      R: KI+ )
       AL cpLOADB       ( P: Suc KIP TVE      R: )
       SWAP             ( P: KIP Suc TVE      R: KI+ )
       AL cpSTOREB      ( P: TVE              R: KI+ )
       DUPL             ( P: TVE TVE          R: KI+ )
       RTOP             ( P: KI+ TVE TVE      R: )
       DUPL             ( P: KI+ KI+ TVE TVE  R: )
       RTD              ( P: TVE KI+ KI+ TVE  R: )
       P0 SUB           ( P: Dif KI+ TVE      R: )
       DEL              ( P: KI+ TVE          R: )
( branch to Loop: )
    -28 NE cBR          ( P: KI+ TVE          R: )

    DEL                 ( P: TVE              R: )
    32 uNLOAD           ( P: 32  TVE          R: )
    SWAP                ( P: TVE 32           R: )
    AL cpSTOREB         ( P:                  R: )
```

```
( branch to PutBlob: )
    104 AL cBR            ( P:                R: )

( Other: ELSE BEGIN )
( IF InsertON AND TV8<=INPUTPOINTER THEN  )

    $$inson AL cLOADB     ( P: Ins Chr        R: )
    01 uN AND             ( P: Ins Chr        R: )
    DEL                   ( P: Chr            R: )
( branch to PutChr: )
    60 EQ cBR             ( P:                R: )

    $$kbdip AL cLOAD      ( P: KIP Chr        R: )
    $$tv8 AL cNLOAD       ( P: TV8 KIP Chr    R: )
    P0 SUB                ( P: Dif Chr        R: )
    DEL                   ( P: Chr            R: )
( branch to PutChr: )
    46 LT cBR             ( P: Chr            R: )

(   FOR Index := TVE DOWNTO INPUTPOINTER+1 DO BEGIN )
    PTOR                  ( P:              R: Chr )
    $$kbdip AL cLOAD      ( P: KIP          R: Chr )
    01 uN ADD             ( P: KI+          R: Chr )
    $$tve AL cNLOAD       ( P: TVE KI+      R: Chr )

( Loop: )
    DUPL                  ( P: TVE TVE KI+      R: Chr )
    01 uN SUB             ( P: TV- TVE KI+      R: Chr )
    DUPL                  ( P: TV- TV- TVE KI+  R: Chr )
    PTOR                  ( P: TV- TVE KI+      R: TV- Chr )
    AL cpLOADB            ( P: Pre TVE KI+      R: TV- Chr )
    SWAP                  ( P: TVE Pre KI+      R: TV- Chr )
    AL cpSTOREB           ( P: KI+              R: TV- Chr )
    DUPL                  ( P: KI+ KI+          R: TV- Chr )
    RTOP                  ( P: TV- KI+ KI+      R: Chr )
    DUPL                  ( P: TV- TV- KI+ KI+  R: Chr )
    RTD                   ( P: KI+ TV- TV- KI+  R: Chr )
    P0 SUB                ( P: Dif TV- KI+      R: Chr )
    DEL                   ( P: TV- KI+          R: Chr )
( branch to Loop: )
    -28 NE cBR            ( P: TV- KI+          R: Chr )
    DEL                   ( P: KI+              R: Chr )
    DEL                   ( P:                  R: Chr )
    RTOP                  ( P: Chr              R: )

( PutChr: )
    $$kbdip AL cLOAD      ( P: KIP Chr          R: )
    DUPL                  ( P: KIP KIP Chr      R: )
    RTU                   ( P: KIP Chr KIP      R: )
    AL cpSTOREB           ( P: KIP              R: )

    01 uN ADD             ( P: KIP              R: )
```

```
    #100 AL cNLOAD        ( P: 100 KIP          R: )
    DUPL                  ( P: 100 100 KIP      R: )
    RTD                   ( P: KIP 100 100      R: )
    P1 RSUB               ( P: KIP 100          R: )
    #3FF AL cNLOAD        ( P: 3FF KIP 100      R: )
    P1 AND                ( P: KIP 100          R: )
    P1 ADD                ( P: KIP              R: )
    $$kbdip AL cSTORE     ( P:                  R: )

( PutBlob:                                        )
( Put Blob Cursor On Screen iff not end of input )
( ---------------------------------------------- )

    $$kbdip AL cLOAD      ( P: KIP              R: )
    $$pe AL cLOAD         ( P: PE  KIP          R: )
    P0 SUB                ( P: Dif              R: )
    DEL                   ( P:                  R: )
( branch to doit: )
    08 HS cBR             ( P:                  R: )
    #01 uNLOAD            ( P: #01              R: )
    $$readyflag AL cSTOREB
    PC POPRS

( doit: )
    $$kbdip AL cLOAD      ( P: KIP              R: )
    DUPL                  ( P: KIP KIP          R: )
    AL cpLOADB            ( P: Chr KIP          R: )
    #80 uN OR             ( P: Chr KIP          R: )
    SWAP                  ( P: KIP Chr          R: )
    AL cpSTOREB           ( P:                  R: )
    PC POPRS

sJSRcomplete ( SU3 StopWatch )
sJSRcomplete ( SU2 StopWatch )
sJSRcomplete ( SU1 StopWatch )
sJSRcomplete ( SU0 StopWatch )

( STOPWATCH )         ( P: SU+0           R: RetAdr )
    DUPL              ( P: SU+0 SU+0      R: RetAdr )
    AL cpLOADB        ( P: mSec SU+0      R: RetAdr )
( if LSB is set, then timer has already expired )
    #1 uN TST         ( P: mSec SU+0      R: RetAdr )
    #48 EQ cBR        ( P: mSec SU+0      R: RetAdr )
( stopwatch has expired, clean stack and exit )
    DEL               ( P: SU+0           R: RetAdr )
    DEL               ( P:                R: RetAdr )
    PC POPRS

( timer not expired, check if mSec is 0 )
    #0 uN CMP         ( P: mSec SU+0      R: RetAdr )
    #08 EQ cBR        ( P: mSec SU+0      R: RetAdr )
( mSec not expired, decrement and exit )
```

```
    #2 uN SUB         ( P: mSec SU+0      R: RetAdr )
    SWAP              ( P: SU+0 mSec      R: RetAdr )
    AL cpSTOREB       ( P:                R: RetAdr )
    PC POPRS

( mSec at 0, reload and check Sec )
    98 uN ADD         ( P: 98   SU+0      R: RetAdr )
    SOT               ( P: SU+0 98   SU+0 R: RetAdr )
    AL cpSTOREB       ( P: SU+0           R: RetAdr )
    #1 uN ADD         ( P: SU+1           R: RetAdr )
    DUPL              ( P: SU+1 SU+1      R: RetAdr )
    AL cpLOADB        ( P: Sec  SU+1      R: RetAdr )
( Sec 0 ? )
    #0 uN CMP         ( P: Sec  SU+1      R: RetAdr )
    #08 EQ cBR        ( P: Sec  SU+1      R: RetAdr )

( Sec not 0, decrement and exit )
    #1 uN SUB         ( P: Sec  SU+1      R: RetAdr )
    SWAP              ( P: SU+1 Sec       R: RetAdr )
    AL cpSTOREB       ( P:                R: RetAdr )
    PC POPRS
( Sec at 0, reload and check min )
    59 uN ADD         ( P: 59   SU+1      R: RetAdr )
    SOT               ( P: SU+1 59   SU+1 R: RetAdr )
    AL cpSTOREB       ( P: SU+1           R: RetAdr )
    #1 uN ADD         ( P: SU+2           R: RetAdr )
    DUPL              ( P: SU+2 SU+2      R: RetAdr )
    AL cpLOAD         ( P: min  SU+2      R: RetAdr )
( Min  0 ? )
    #0 uN CMP         ( P: min  SU+2      R: RetAdr )
    #08 EQ cBR        ( P: min SU+2       R: RetAdr )
( Min not 0, decrement and exit)
    #1 uN SUB         ( P: min  SU+2      R: RetAdr )
    SWAP              ( P: SU+0 min       R: RetAdr )
    AL cpSTORE        ( P:                R: RetAdr )
    PC POPRS

( timer just expired, clean up and set expired flag in mSec )
    DEL               ( P: SU+2           R: RetAdr )
    #2 uN SUB         ( P: SU+0           R: RetAdr )
    #1 uNLOAD         ( P: 1    SU+0      R: RetAdr )
    SWAP              ( P: SU+0 1         R: RetAdr )
    AL cpSTOREB       ( P:                R: RetAdr )
    PC POPRS

sJSRcomplete ( UHR )

#0518 AL cNLOAD      ( P: UHR+0             R: RetAdr )
    DUPL             ( P: UHR+0 UHR+0       R: RetAdr )
    AL cpLOADB       ( P: mSec  UHR+0       R: RetAdr )
( if mSec is 98, then update and check Sec )
    98 uN CMP        ( P: mSec  UHR+0       R: RetAdr )
```

```
    #08 EQ cBR       ( P: mSec  UHR+0       R: RetAdr )
( else inc mSec by 2 and exit )
    #2 uN ADD        ( P: mSec  UHR+0       R: RetAdr )
    SWAP             ( P: UHR+0 mSec        R: RetAdr )
    AL cpSTOREB
    PC POPRS

( update mSec and check Sec )
    DEL              ( P: 0                 R: RetAdr )
    #0 uNLOAD        ( P: 0     UHR+0       R: RetAdr )
    SOT              ( P: UHR+0     0 UHR+0 R: RetAdr )
    AL cpSTOREB      ( P: UHR+0             R: RetAdr )
    #1 uN ADD        ( P: UHR+1             R: RetAdr )
    DUPL             ( P: UHR+1 UHR+1       R: RetAdr )
    AL cpLOADB       ( P: Sec   UHR+1       R: RetAdr )
( if Sec is 59, then update and check Min )
    59 uN CMP        ( P: Sec   UHR+1       R: RetAdr )
    #08 EQ cBR       ( P: Sec   UHR+1       R: RetAdr )
( else inc Sec by 1 and exit )
    #1 uN ADD        ( P: Sec   UHR+1       R: RetAdr )
    SWAP             ( P: UHR+1 Sec         R: RetAdr )
    AL cpSTOREB
    PC POPRS

( update Sec and check Min )
    DEL              ( P:                   R: RetAdr )
    #0 uNLOAD        ( P: 0     UHR+1       R: RetAdr )
    SOT              ( P: UHR+1     0 UHR+1 R: RetAdr )
    AL cpSTOREB      ( P: UHR+1             R: RetAdr )
    #1 uN ADD        ( P: UHR+2             R: RetAdr )
    DUPL             ( P: UHR+2 UHR+2       R: RetAdr )
    AL cpLOADB       ( P: Min   UHR+2       R: RetAdr )
( if Min is 59, then update and check Hour )
    59 uN CMP        ( P: Min   UHR+2       R: RetAdr )
    #08 EQ cBR       ( P: Min   UHR+2       R: RetAdr )
( else inc Min by 1 and exit )
    #1 uN ADD        ( P: Min   UHR+2       R: RetAdr )
    SWAP             ( P: UHR+2 Min         R: RetAdr )
    AL cpSTOREB
    PC POPRS

( update Min and check Hour )
    DEL              ( P:                   R: RetAdr )
    #0 uNLOAD        ( P: 0     UHR+2       R: RetAdr )
    SOT              ( P: UHR+2     0 UHR+2 R: RetAdr )
    AL cpSTOREB      ( P: UHR+2             R: RetAdr )
    #1 uN ADD        ( P: UHR+3             R: RetAdr )
    DUPL             ( P: UHR+3 UHR+3       R: RetAdr )
    AL cpLOADB       ( P: Hour  UHR+3       R: RetAdr )
( if Hour is 23, then update and inc Day )
    23 uN CMP        ( P: Hour  UHR+3       R: RetAdr )
    #08 EQ cBR       ( P: Hour  UHR+3       R: RetAdr )
```

```
( else inc Hour by 1 and exit )
    #1 uN ADD        ( P: Hour  UHR+3       R: RetAdr )
    SWAP             ( P: UHR+3 Hour        R: RetAdr )
    AL cpSTOREB
    PC POPRS

( update Hour and increment Day )
    DEL
    #0 uNLOAD        ( P: 0     UHR+3       R: RetAdr )
    SOT              ( P: UHR+3     0 UHR+3 R: RetAdr )
    AL cpSTOREB      ( P: UHR+3             R: RetAdr )
    #1 uN ADD        ( P: UHR+4             R: RetAdr )
    DUPL             ( P: UHR+4 UHR+4       R: RetAdr )
    AL cpLOAD        ( P: Days  UHR+4       R: RetAdr )
    #1 uN ADD        ( P: Days  UHR+4       R: RetAdr )
    AL cpSTORE
    PC POPRS

(                      Code Routines                          )
(                      -------------                          )

( DEFEX  ) hier $ccodes !n DFX NEXT
( VAREX  ) hier $ccodes 02 +n !n HP PUSHPS NEXT
( CONSEX ) hier $ccodes 04 +n !n HP PUSHPS AL cpLOAD NEXT

code RUMPELSTILZCHEN NEXT
code RETEX PPC POPRS NEXT

code $JEEX         ( P Index Limit        R - )
( enter with limit and index on P stack )
    JPPC           ( P Index Limit        R - )
    PTOR           ( P Index              R Limit )
    IDX            ( P Index Limit        R Limit )
    P1 RCMP        ( P Index Limit        R Limit )
( IF I<=L THEN )   (Limit-Index)
    HI cBRbegin    ( P Index Limit        R Limit )
        SWAP       ( P Limit Index        R Limit )
        PTOR       ( P Limit              R Limit Index )
        DEL        ( P                    R Limit Index )
        JPPC       ( P                    R Limit Index )
( exit with index and limit on R )
    cBRelse
        RTOP       ( P Index Limit Limit  R - )
        DEL        ( P Index Limit        R - )
        DEL        ( P Index              R - )
        DEL        ( P -                  R - )
        PPC PUSHPS ( P PPC                R - )
        #2 uN ADD  ( P PPC+2              R - )
        PPC POPPS  ( P -                  R - )
( exit with stacks empty and PPC pointing to next word )
    cBRend
```

```
    NEXT

code +LOOPEX       ( P Inc                R Limit Index )
    RTOP           ( P Inc Index          R Limit       )
    P1 ADD         ( P Index              R Limit )
    IDX            ( P Index Limit        R Limit )
    P1 RCMP        ( P Index Limit        R Limit )
( IF I<=L THEN )
    HI cBRbegin    ( P Index Limit        R Limit )
        SWAP       ( P Limit Index        R Limit )
        PTOR       ( P Limit              R Limit Index )
        DEL        ( P                    R Limit Index )
        JPPC       ( P                    R Limit Index )
( exit with index and limit on R )
    cBRelse
        RTOP       ( P Index Limit Limit  R - )
        DEL        ( P Index Limit        R - )
        DEL        ( P Index              R - )
        DEL        ( P -                  R - )
        PPC PUSHPS ( P PPC                R - )
        #2 uN ADD  ( P PPC+2              R - )
        PPC POPPS  ( P -                  R - )
( exit with stacks empty and PPC pointing to next word )
    cBRend
   NEXT

code LOOPEX        ( P -                  R Limit Index )
    RTOP           ( P Index              R Limit )
    #01 uN ADD     ( P Index              R Limit )
    IDX            ( P Index Limit        R Limit )
    P1 RCMP        ( P Index Limit        R Limit )
( IF I<=L THEN )
    HI cBRbegin    ( P Index Limit        R Limit )
        SWAP       ( P Limit Index        R Limit )
        PTOR       ( P Limit              R Limit Index )
        DEL        ( P                    R Limit Index )
        JPPC       ( P                    R Limit Index )
( exit with index and limit on R )
    cBRelse
        RTOP       ( P Index Limit Limit  R - )
        DEL        ( P Index Limit        R - )
        DEL        ( P Index              R - )
        DEL        ( P -                  R - )
        PPC PUSHPS ( P PPC                R - )
        #2 uN ADD  ( P PPC+2              R - )
        PPC POPPS  ( P -                  R - )
( exit with stacks empty and PPC pointing to next word )
    cBRend
    NEXT

code 2BLITERAL 2BLIT NEXT
```

```
code BRONZ
    #01 uN AND DEL NE cBRbegin
        JPPC
    cBRelse
        PPC PUSHPS
        #2 uN ADD
        PPC POPPS
    cBRend
    NEXT

code @     AL cpLOAD  NEXT
code @B    AL cpLOADB NEXT
code !     AL cpSTORE NEXT
code !B    AL cpSTORE NEXT
code JUMP  JPPC       NEXT
code +     P1 ADD     NEXT
code -     P0 SUB     NEXT
code NICHT CPL        NEXT
code UND   P1 AND     NEXT
code ODER  P1 OR      NEXT
code EXO   P1 EOR     NEXT
code BIT   P0 MASK    NEXT
code CHS   NEG        NEXT
code WEG   DEL        NEXT
code PWEG  DEL DEL    NEXT
code DUP   DUPL       NEXT
code PDUP  SOT SOT    NEXT
code VERT  SWAP       NEXT
code ZWO   SOT        NEXT
code RDU   RTU        NEXT
code RDO   RTD        NEXT
code I     IDX        NEXT
code S>R   PTOR       NEXT
code R>S   RTOP       NEXT

code =0  #0 sN CMP EQ FLAG NEXT
code <0  #0 sN CMP LT FLAG NEXT
code >0  #0 sN CMP GT FLAG NEXT
code >=U    P0 SUB LS FLAG NEXT

code F-VERGL     ( Field Compare, Unsigned, 1 to 256 bytes )
                 ( P: n  a2 a1        R: )
( assume fields are equal, set t=1 for initial comparison )
    #1 uNLOAD    ( P: t  n  a2 a1     R: )
    SWAP         ( P: n  t a2 a1      R: )

( a: )
    PTOR         ( P: t a2 a1         R: n )
    PTOR         ( P: a2 a1           R: t n )
    DUPL         ( P: a2 a2 a1        R: t n )
    AL cpLOADB   ( P: <a2> a2 a1      R: t n )
    RTD          ( P: a1 <a2> a2      R: t n )
```

```
    DUPL          ( P: a1 a1 <a2> a2   R: t n )
    AL cpLOADB    ( P: <a1> a1 <a2> a2 R: t n )
    RTD           ( P: <a2> <a1> a1 a2 R: t n )
    P0 SUB        ( P: dif a1 a2       R: t n )
    DEL           ( P: a1 a2           R: t n )

( if they are equal, skip further testing for this pair )
( banch to label c: )
    #0C EQ cBR    ( P: a1 a2           R: t n )

( they are not equal, so t must be updated )
    RTOP          ( P: t a1 a2         R: n )
    DEL           ( P: a1 a2           R: n )
    #0 uNLOAD     ( P: t a1 a2         R: n )

( branch to label b: if a1<a2, else t=0 )
    #02 CS cBR    ( P: t a1 a2         R: n )

( a1>a2, t=2 )
    #2 uN ADD     ( P: t a1 a2         R: n )
    PTOR          ( P: a1 a2           R: t n )

( c: equal or t updated )
    #1 uNLOAD     ( P:  1 a1 a2        R: t n )
    P1 ADD        ( P: a1+ a2          R: t n )
    SWAP          ( P: a2  a1+         R: t n )
    #1 uNLOAD     ( P:  1 a2 a1+       R: t n )
    P1 ADD        ( P: a2+ a1+         R: t n )
    RTOP          ( P: t a2+ a1+       R: n )
    RTOP          ( P: n t a2+ a1+     R: )
    #1 uN SUB     ( P: n t a2+ a1+     R: )
    #FF uN AND    ( P: n t a2+ a1+     R: )

( branch to label a: if all elements not checked)
    #CA NE cBR    ( P: n t a2+ a1+     R: )

( done, clean up stack and exit )
    DEL           ( P: t a2+ a1+       R: )
    RTU           ( P: a2+ a1+ t       R: )
    DEL           ( P: a1+ t           R: )
    DEL           ( P: t               R: )

    NEXT          ( P: t               R: )

code SBIT
    AL cpLOAD
    SWAP
    P0 MASK
    P1 OR
    NEXT

code CBIT
```

```
    AL cpLOAD
    SWAP
    P0 MASK
    CPL
    P1 AND
    NEXT

code TBIT
    AL cpLOAD
    SWAP
    P0 MASK
    P1 AND
    NE FLAG
    NEXT

code >>>        (Field Transport up to 256 bytes )
                ( P: b ad as        R: )
( a: )
    PTOR        ( P: ad as          R: b )
    DUPL        ( P: ad ad as       R: b )
    RTD         ( P: as ad ad       R: b )
    DUPL        ( P: as as ad ad    R: b )
    AL cpLOADB  ( P: <as> as ad ad  R: b )
    RTD         ( P: ad <as> as ad  R: b )
    AL cpSTOREB ( P: as ad          R: b )
    #1 uN ADD   ( P: as ad          R: b )
    SWAP        ( P: ad as          R: b )
    #1 uN ADD   ( P: ad as          R: b )
    RTOP        ( P: b ad as        R: )
    #1 uN SUB   ( P: b ad as        R: )
    #FF uN AND  ( P: b ad as        R: )
( branch to label a: )
    #E4 NE cBR  ( P: b ad as        R: )
    DEL         ( P: ad as          R: )
    DEL         ( P: as             R: )
    DEL         ( P:                R: )

    NEXT

code $TUE
    HP POPPS
    AL EXECUTE

code $IPSETZEN
    $$kbdip sSTORE
    DEL
    NEXT

code $PSHOLEN
    PSC PUSHPS
    NEXT
```

```
( this routine assumes that it is to be used to clear or  )
( reset the stack to empty, after a call to CLS or by an  )
( underflow in the compiler. The overflow area is always  )
( reset on each call, ... so take care using this routine )
( for anything else!                                      )

code $PSSETZEN
    PSC POPPS
    #FEFE AL cNLOAD PSP POPPS                  ( Set PSP )
    NEXT

( 32-bit add )
( expects the two values to be added to be present on the )
( stack, high byte on top )
( returns the 32-bit result, high byte on top )

code P+         ( A + B )
                ( P: Bh Bl Ah Al R: )
    PTOR        ( P: Bl Ah Al    R: Bh )
    RTD         ( P: Al Bl Ah    R: Bh )
    P1 ADD      ( P: Cl Ah       R: Bh )
    RTOP        ( P: Bh Cl Ah    R: )
    RTD         ( P: Ah Bh Cl    R: )
    P1 ADC      ( P: Ch Cl       R: )
    NEXT

( 32-bit subtract )
( Expects the subtrahend on the top of the stack and the )
( minuend below it )
( The difference, minuend - subtrahend,  is returned on )
( the top of the stack, high byte on top )

code P-         ( A - B )
                ( P: Bh Bl Ah Al R: )
    PTOR        ( P: Bl Ah Al    R: Bh )
    RTD         ( P: Al Bl Ah    R: Bh )
    P0 SUB      ( P: Cl Ah       R: Bh )
    RTOP        ( P: Bh Cl Ah    R: )
    RTD         ( P: Ah Bh Cl    R: )
    P0 SBC      ( P: Ch Cl       R: )
    NEXT

code P*         ( A * B )
                ( P: Al Bl          R: )
    PTOR        ( P: Bl             R: Al )
    #0 uNLOAD   ( P: Bh Bl          R: Al )
    DUPL        ( P: Cl Bh Bl       R: Al )
    DUPL        ( P: Ch Cl Bh Bl    R: Al )
( a: )
    RTOP        ( P: Al Ch Cl Bh Bl R: )
    #0 uN CMP   ( P: Al Ch Cl Bh Bl R: )
    #2C EQ cBR   ( P: Al Ch Cl Bh Bl R: )
```

```
    ( branch to label c: )

    LSR         ( P: Al Ch Cl Bh Bl R: )
    PTOR        ( P: Ch Cl Bh Bl    R: Al )
    #14 CC cBR  ( P: Ch Cl Bh Bl    R: Al )
    ( branch to label b: )

    PTOR        ( P: Cl Bh Bl       R: Ch Al )
    PTOR        ( P: Bh Bl          R: Cl Ch Al )
    SOT         ( P: Bl Bh Bl       R: Cl Ch Al )
    SOT         ( P: Bh Bl Bh Bl    R: Cl Ch Al )
    RTOP        ( P: Cl Bh Bl Bh Bl R: Ch Al )
    RTD         ( P: Bl Cl Bh Bh Bl R: Ch Al )
    P1 ADD      ( P: Cl Bh Bh Bl    R: Ch Al )
    RTOP        ( P: Ch Cl Bh Bh Bl R: Al )
    RTD         ( P: Bh Ch Cl Bh Bl R: Al )
    P1 ADC      ( P: Ch Cl Bh Bl    R: Al )
( b: )
    PTOR        ( P: Cl Bh Bl       R: Ch Al )
    PTOR        ( P: Bh Bl          R: Cl Ch Al )
    SWAP        ( P: Bl Bh          R: Cl Ch Al )
    LSL         ( P: Bl Bh          R: Cl Ch Al )
    SWAP        ( P: Bh Bl          R: Cl Ch Al )
    ROL         ( P: Bh Bl          R: Cl Ch Al )
    RTOP        ( P: Cl Bh Bl       R: Ch Al )
    RTOP        ( P: Ch Cl Bh Bl    R: Al )
    #CE AL cBR  ( P: Ch Cl Bh Bl    R: Al )
    ( branch to label a: )
( c: )
    DEL         ( P: Ch Cl Bh Bl    R: )
    PTOR        ( P: Cl Bh Bl       R: Ch )
    PTOR        ( P: Bh Bl          R: Cl Ch )
    DEL         ( P: Bl             R: Cl Ch )
    DEL         ( P:                R: Cl Ch )
    RTOP        ( P: Cl             R: Ch )
    RTOP        ( P: Ch Cl          R: )

    NEXT

code P/MOD              ( P: D  Nh Nl                R: )
( align divisor with quotient )
    #0 uNLOAD       ( P: Dl Dh Nh Nl             R: )
    SWAP            ( P: Dh Dl Nh Nl             R: )
    SOT             ( P: pl Dh Dl Nh Nl          R: )
( place marker )

    #1 uNLOAD       ( P: ph pl Dh Dl Nh Nl       R: )
( initialize quotient )
    #0 uNLOAD       ( P: ql ph pl Dh Dl Nh Nl    R: )
    DUPL            ( P: qh ql ph pl Dh Dl Nh Nl R: )
    PTOR            ( P: ql ph pl Dh Dl Nh Nl R: qh )
    PTOR            ( P: ph pl Dh Dl Nh Nl R: ql qh )
```

```
    PTOR            ( P: pl Dh Dl Nh Nl R: ph ql qh )
    PTOR            ( P: Dh Dl Nh Nl R: pl ph ql qh )
( repeat: )
( if N>=D )
    RTD             ( P: Nh Dh Dl Nl R: pl ph ql qh )
    P0 CMP          ( P: Nh Dh Dl Nl R: pl ph ql qh )
( branch to qnupdate )
    #0E HI cBR      ( P: Nh Dh Dl Nl R: pl ph ql qh )
    #48 LO cBR      ( P: Nh Dh Dl Nl R: pl ph ql qh )
    PTOR            ( P: Dh Dl Nl R: Nh pl ph ql qh )
    PTOR            ( P: Dl Nl R: Dh Nh pl ph ql qh )
    P1 RCMP         ( P: Dl Nl R: Dh Nh pl ph ql qh )
    RTOP            ( P: Dh Dl Nl R: Nh pl ph ql qh )
    RTOP            ( P: Nh Dh Dl Nl R: pl ph ql qh )
    #3C LO cBR      ( P: Nh Dh Dl Nl R: pl ph ql qh )
( branch to shift )
( qnupdate: )
    RTOP            ( P: pl Nh Dh Dl Nl R: ph ql qh )
    DUPL            ( P: pl pl Nh Dh Dl Nl R: ph ql qh )
    RTOP            ( P: ph pl pl Nh Dh Dl Nl R: ql qh )
    SWAP            ( P: pl ph pl Nh Dh Dl Nl R: ql qh )
    RTOP            ( P: ql pl ph pl Nh Dh Dl Nl R: qh )
    P0 ADD          ( P: ql ph pl Nh Dh Dl Nl R: qh )
    SWAP            ( P: ph ql pl Nh Dh Dl Nl R: qh )
    DUPL            ( P: ph ph ql pl Nh Dh Dl Nl R: qh )
    RTOP            ( P: qh ph ph ql pl Nh Dh Dl Nl R: )
    P0 ADC          ( P: qh ph ql pl Nh Dh Dl Nl R: )
    PTOR            ( P: ph ql pl Nh Dh Dl Nl R: qh )
    SWAP            ( P: ql ph pl Nh Dh Dl Nl R: qh )
    PTOR            ( P: ph pl Nh Dh Dl Nl R: ql qh )
    PTOR            ( P: pl Nh Dh Dl Nl R: ph ql qh )
    PTOR            ( P: Nh Dh Dl Nl R: pl ph ql qh )
    RTU             ( P: Dh Dl Nh Nl R: pl ph ql qh )
    SOT             ( P: Dl Dh Dl Nh Nl R: pl ph ql qh )
    SOT             ( P: Dh Dl Dh Dl Nh Nl R: pl ph ql qh )
    PTOR            ( P: Dl Dh Dl Nh Nl R: Dh pl ph ql qh )
    PTOR            ( P: Dh Dl Nh Nl R: Dl Dh pl ph ql qh )
    PTOR            ( P: Dl Nh Nl R: Dh Dl Dh pl ph ql qh )
    RTD             ( P: Nl Dl Nh R: Dh Dl Dh pl ph ql qh )
    P1 RSUB         ( P: Nl Nh R: Dh Dl Dh pl ph ql qh )
    RTOP            ( P: Dh Nl Nh R: Dl Dh pl ph ql qh )
    RTD             ( P: Nh Dh Nl R: Dl Dh pl ph ql qh )
    P1 RSBC         ( P: Nh Nl R: Dl Dh pl ph ql qh )
    RTOP            ( P: Dl Nh Nl R: Dh pl ph ql qh )
    SWAP            ( P: Nh Dl Nl R: Dh pl ph ql qh )
    RTOP            ( P: Dh Nh Dl Nl R: pl ph ql qh )
    SWAP            ( P: Nh Dh Dl Nl R: pl ph ql qh )

( shift: )
    SWAP            ( P: Dh Nh Dl Nl R: pl ph ql qh )
    LSR             ( P: Dh Nh Dl Nl R: pl ph ql qh )
    RTD             ( P: Dl Dh Nh Nl R: pl ph ql qh )
```

```
    ROR             ( P: Dl Dh Nh Nl R: pl ph ql qh )
    SWAP            ( P: Dh Dl Nh Nl R: pl ph ql qh )
    RTOP            ( P: pl Dh Dl Nh Nl R: ph ql qh )
    RTOP            ( P: ph pl Dh Dl Nh Nl R: ql qh )
    LSR             ( P: ph pl Dh Dl Nh Nl R: ql qh )
    PTOR            ( P: pl Dh Dl Nh Nl R: ph ql qh )
    ROR             ( P: pl Dh Dl Nh Nl R: ph ql qh )
( done? branch to chkerr )
    #04 EQ cBR      ( P: pl Dh Dl Nh Nl R: ph ql qh )
    PTOR            ( P: Dh Dl Nh Nl R: pl ph ql qh )
( not done - branch to repeat )
    #96 AL cBR      ( P: Dh Dl Nh Nl R: pl ph ql qh )

( chkerr: )
    DEL             ( P: Dh Dl Nh Nl R: ph ql qh )
    DEL             ( P: Dl Nh Nl R: ph ql qh )
    DEL             ( P: Nh Nl R: ph ql qh )
    RTOP            ( P: ph Nh Nl R: ql qh )
    DEL             ( P: Nh Nl R: ql qh )
    RTOP            ( P: ql Nh Nl R: qh )
    RTOP            ( P: qh ql Nh Nl R: )
    #0 uN RSUB      ( P: qh ql Nh Nl R: )
    DEL             ( P: ql Nh Nl R: )
( branch to error )
    #0A NE cBR      ( P: ql Nh Nl R: )
( OK: )
    SWAP            ( P: Nh ql Nl R: )
    DEL             ( P: ql Nl R: )
    SWAP            ( P: Nl ql R: )

    NEXT            ( P: rem quo )

( error: )
    DEL             ( P: Nh Nl R: )
    DEL             ( P: Nl R: )
    DEL             ( P: R: )
    #FF sNLOAD      ( P: quo )
    #0 uNLOAD       ( P: rem quo )

    NEXT            ( P: rem quo )


code $POLYNAME ( P: ch      0.C     B.A         R: )

    PTOR       ( P: 0.C     B.A                 R: ch )
    SWAP       ( P: B.A     0.C                 R: ch )
    DUPL       ( P: B.A     B.A     0.C         R: ch )
    #FF uN AND ( P: 0.A     B.A     0.C         R: ch )
    RTU        ( P: B.A     0.C     0.A         R: ch )
    XB         ( P: A.B     0.C     0.A         R: ch )
    #FF uN AND ( P: 0.B     0.C     0.A         R: ch )
    XB         ( P: B.0     0.C     0.A         R: ch )
```

```
    P1 OR       ( P: B.C      0.A                 R: ch )
    SWAP        ( P: 0.A      B.C                 R: ch )
    SOT         ( P: B.C      0.A     B.C         R: ch )
    DUPL        ( P: B.C      B.C     0.A B.C     R: ch )
    LSR         ( P: B>1.C>1  B.C     0.A B.C     R: ch )
    DUPL        ( P: B>1.C>1  B>1.C>1 B.C 0.A B.C R: ch )
    LSR         ( P: B>2.C>2  B>1.C>1 B.C 0.A B.C R: ch )
    P1 EOR      ( P: BxC B.C  0.A     B.C         R: ch )
    P1 EOR      ( P: BxC 0.A  B.C                 R: ch )
    PTOR        ( P: 0.A B.C                      R: BxC ch )
    SOT         ( P: B.C 0.A  B.C                 R: BxC ch )
    SOT         ( P: 0.A B.C  0.A     B.C         R: BxC ch )
    SWAP        ( P: B.C 0.A  0.A     B.C         R: BxC ch )
    LSL         ( P: B<1.C<1  0.A 0.A B.C         R: BxC ch )
    SWAP        ( P: 0.A      B<1.C<1 0.A B.C     R: BxC ch )
    ROL         ( P: 0<1.A<1  B<1.C<1 0.A B.C     R: BxC ch )
    #FF uN AND  ( P: 0.A<1    B<1.C<1 0.A B.C     R: BxC ch )
    XB          ( P: A.0<1    B<1.C<1 0.A B.C     R: BxC ch )
    SWAP        ( P: B<1.C<1  A.0<1   0.A B.C     R: BxC ch )
    XB          ( P: C<1.B<1  A.0<1   0.A B.C     R: BxC ch )
    #FF uN AND  ( P: 0.B<1    A.0<1   0.A B.C     R: BxC ch )
    P1 OR       ( P: A<1.B<1  0.A     B.C         R: BxC ch )
    RTOP        ( P: BxC      A<1.B<1 0.A B.C     R: ch )
    P1 EOR      ( P: BxC      0.A     B.C         R: ch )
    RTOP        ( P: ch       BxC     0.A B.C     R: )
    #FF uN AND  ( P: ch       BxC     0.A B.C     R: )
    P1 EOR      ( P: BxC      0.A     B.C         R: )
    #FF uN AND  ( P: 0.xC     0.A     B.C         R: )
    XB          ( P: xC.0     0.A     B.C         R: )
    P1 OR       ( P: xC.A     B.C                 R: )
    SWAP        ( P: B.C      xC.A                R: )
    LSL         ( P: B<1.C<1  xC.A                R: )
    SWAP        ( P: xC.A     B<1.C<1             R: )
    ROL         ( P: xC<1.A<1 B<1.C<1             R: )
    XB          ( P: A<1.xC<1 B<1.C<1             R: )
    SWAP        ( P: B<1.C<1  A<1.xC<1            R: )
    XB          ( P: C<1.B<1  A<1.xC<1            R: )
    #FF uN AND  ( P: 0.B<1    A<1.xC<1            R: )

    NEXT

code CYC2           ( P: 0.ch A.B       R: )
    #FF uN AND      ( P: 0.ch A.B       R: )
    XB              ( P: ch.0 A.B       R: )
    SWAP            ( P: A.B  ch.0      R: )
    XB              ( P: B.A  ch.0      R: )
    #8 uNLOAD       ( P: cnt B.A  ch.0  R: )

( a: LOOP )
    PTOR            ( P: B.A  ch.0      R: cnt )
    SOT             ( P: ch.0 B.A  ch.0 R: cnt )
    P1 EOR          ( P: cXba ch.0      R: cnt )
```

```
( set flag if MSB 1 )
    MI FLAG          ( P: 0|1  cXba ch.0  R: cnt )
    PTOR             ( P: cXba ch.0       R: 0|1 cnt )
    LSL              ( P: cXba ch.0       R: 0|1 cnt )
    SWAP             ( P: ch.0 cXba       R: 0|1 cnt )
    LSL              ( P: ch.0 cXba       R: 0|1 cnt )
    SWAP             ( P: cXba ch.0       R: 0|1 cnt )
    RTOP             ( P: 0|1  cXba ch.0  R: cnt )
    #1 uN AND        ( P: 0|1  cXba ch.0  R: cnt )
    DEL              ( P: cXba ch.0       R: cnt )

( branch to label b if MSB 0 )
    #6 EQ cBR        ( P: cXba ch.0       R: cnt )
    #1021 AL cNLOAD  ( P: 1021 cXba ch.0  R: cnt )
    P1 EOR           ( P: cXba ch.0       R: cnt )

( b: test LOOP CNT to see if we are done)
    RTOP             ( P: cnt cXba ch.0   R: )
    #1 uN SUB        ( P: cnt cXba ch.0   R: )

( repeat loop if cnt <> 0 )
    #DA NE cBR       ( P: cnt cXba ch.0   R: )

( tidy up and return result in correct byte order )
    RTU              ( P: ch.0 cnt cXba   R: )
    DEL              ( P: cnt cXba        R: )
    DEL              ( P: cXba            R: )
    XB               ( P: A.B             R: )

    NEXT             ( P: A.B             R: )

code TR-LOOP                             ( not required for now )
( ** TODO ** )
    NEXT

code RP-LOOP                             ( not required for now )
( ** TODO ** )
    NEXT

code 3V3  ( P: A B C D E F   R: )
    PTOR  ( P: B C D E F     R: A )
    RTD   ( P: D B C E F     R: A )
    XRP   ( P: A B C E F     R: D )
    PTOR  ( P: B C E F       R: A D )
    RTD   ( P: E B C F       R: A D )
    XRP   ( P: A B C F       R: E D )
    PTOR  ( P: B C F         R: A E D )
    RTD   ( P: F B C         R: A E D )
    XRP   ( P: A B C         R: F E D )
    RTOP  ( P: F A B C       R: E D )
    RTOP  ( P: E F A B C     R: D )
    RTOP  ( P: D E F A B C   R: )
```

```
    NEXT

(                        IPS-F1G                                )
(                        -------                                )

#0518  kon  UHR
#051E  kon  SU0
#0522  kon  SU1
#0526  kon  SU2
#052A  kon  SU3
#0502  kon  KETTE

0 kon 0   1 kon 1   2 kon 2   4 kon 4
'n 2BLITERAL #0 'n BRONZ 'n JUMP 'n $JEEX  'n LOOPEX
'n +LOOPEX   'n RETEX  $ccodes 6 +n 8 !fk

(                   IPS general definitions                     )

:n >    - >0   ;n                :n <>    - =0 NICHT      ;n
:n =    - =0   ;n                :n >=    - <0 NICHT      ;n
:n <    - <0   ;n                :n <=    - >0 NICHT      ;n
                                 :n <>0     =0 NICHT      ;n
:n *       P* WEG ;n             :n P/     P/MOD WEG      ;n
:n /MOD  #0 VERT P/MOD ;n        :n /       /MOD WEG      ;n
:n MOD  /MOD VERT WEG ;n         :n +! DUP @ RDO + VERT ! ;n

(                     Compiler constants                        )

#02C0  kon  SYSLINE              ( Posn.  buffer for messages )
#052E  kon  READYFLAG        ( Compiler free to process input )
#052F  kon  LOADFLAG                 ( Input coming from file )
#0530  kon  $PE                    ( Pointer to end of input )
#0532  kon  $PI                  (     Compiler read pointer )
#0534  kon  $P1                  ( Compiler parsing position )
#0536  kon  $P2                  ( End of block reached flag )
#0538  kon  $P3                    ( Link pointer for $SUCH  )
#053A  kon  $H                  ( Pointer to memory position )

#0004  kon  $SL                                ( Stack limit )
#FF00  kon  $ML                          ( Memory limit 64K )

#0000  kon  $LL                                 ( End of IPS )

#0100  kon  TV0               ( 1st   TV screen line position )
#0200  kon  TV4               ( 4th   TV screen line position )
#0300  kon  TV8               ( 8th   TV screen line position )
#0100  kon  $TVS             ( Stack TV screen line position )
#04FF  kon  $TVE             ( Last  TV screen line position )

$ccodes @n kon DEFEX
$ccodes @n 02 +n kon VAREX
```

```
$ccodes @n 04 +n kon CONSEX

(                        The Compiler                          )
(                        ------------                          )

#0004 feld $ND
#0001 var $RS
#0000 var $F1
#0000 var $F2
#0000 var $KK
#0000 var BASIS
#0000 var BEM
#0001 var BEA
#0000 var EINGABEZAHL
#0000 var Z-LESEN
#0000 var COMPILEFLAG
#0000 var $V1
#0000 var LINK

(                       Error messages                         )
(                       --------------                         )

(  Default language     Alternative language                   )

16 feld STACKMESSAGE    16 feld L-STACKMESSAGE
16 feld MEMMESSAGE      16 feld L-MEMMESSAGE
16 feld NAMEMESSAGE     16 feld L-NAMEMESSAGE
16 feld STRUCMESSAGE    16 feld L-STRUCMESSAGE
16 feld TEXTMESSAGE     16 feld L-TEXTMESSAGE
16 feld RSMESSAGE       16 feld L-RSMESSAGE

~ SPEICHER VOLL !  ~   'n MEMMESSAGE       02 +n $OC !t
~ MEMORY FULL !    ~   'n L-MEMMESSAGE     02 +n $OC !t
~ NAME FEHLT !     ~   'n NAMEMESSAGE      02 +n $OC !t
~ NAME MISSING !   ~   'n L-NAMEMESSAGE    02 +n $OC !t
~ STAPEL LEER !    ~   'n STACKMESSAGE     02 +n $OC !t
~ STACK EMPTY !    ~   'n L-STACKMESSAGE   02 +n $OC !t
~ STRUKTURFEHLER ! ~   'n STRUCMESSAGE     02 +n $OC !t
~ STRUCTURE ERROR! ~   'n L-STRUCMESSAGE   02 +n $OC !t
~ TEXTFEHLER !     ~   'n TEXTMESSAGE      02 +n $OC !t
~ TEXT-ERROR !     ~   'n L-TEXTMESSAGE    02 +n $OC !t
~ UNZUL. NAME !    ~   'n RSMESSAGE        02 +n $OC !t
~ DUPLICATE NAME ! ~   'n L-RSMESSAGE      02 +n $OC !t

(                   Compiler definitions                      )
(                   --------------------                      )

:n INCR  DUP @ 1 + VERT ! ;n

:n HIER  $H @ ;n

:n H2INC HIER 2 + $H ! ;n
```

```
:n $DEP  HIER ! H2INC ;n

:n $CEN  DUP $IPSETZEN DUP @B #80 ODER ZWO !B
         $PI ! $TVE $PE ! 0 READYFLAG !B ;n

:n IE    $P1 @ DUP $PI @ 1 - je I @B #80 EXO I !B
         nun $CEN WEG ;n

#0 kon $LANG                          ( Messages language switch )

:n SYSWRITE $LANG + SYSLINE 16 >>> 0 IE ;n

:n L>>> anfang DUP 256   > ja? 256 - S>R PDUP 256 >>>
                               256 + VERT 256 + VERT R>S
        dann/nochmal DUP >0 ja?  >>>
                             nein: PWEG WEG
                             dann ;n

( :n $SUCH  LINK @ $P3 ! $SCODE ;n )

:n $SUCH  LINK @
   anfang  DUP @B #3F UND $ND @B = ZWO 1 + $ND 1 +
   3 F-VERGL UND NICHT
   ja?  ( NICHT GEF. )  4 + @
      DUP =0 ja?  ( LISTENENDE )  RETEX
             dann
   dann/nochmal 6 + ;n

#0 var CFLAG ( Comment Flag )

:n $CSCAN 0 $PI @ $PE @
   je WEG 1 I @B #20 EXO >0
     ja? I @B CFLAG @B ja? #29 ( KL. ZU ) =
                           ja? 0 CFLAG !B
                           dann
                      nein: #28 ( KL. AUF ) =
                            ja? 1 CFLAG !B
                            nein: WEG 2
                            dann
                      dann
     dann VERT ZWO = ja? 0
                     nein: R>S $PI ! I S>R
                     dann
   nun DUP =0 ja? $PE @ 1 + $PI ! VERT WEG 1 $P2 !
              dann ;n

:n $NAME   0   READYFLAG @B 0 $P2 !
       ja? 1 $CSCAN >0
          ja? $PI @ $P1 !
              2 $CSCAN PWEG #CE57 #8D
              $P1 @ $PI @ ZWO - DUP 63 > ja? WEG 63
```

```
                                          dann
             DUP $ND !B 1 - ZWO +
             je I @B $POLYNAME
             nun $ND 3 + !B $ND 1 + ! 1
          dann
       dann ;n

:n $ZAHL 1 ( OK ) 0 ( ANF. ) $PI @ 1 - $P1 @
  #2D ZWO @B = ja?    1 +  -1 S>R ( NEG ) 10 ( BASIS )
               nein:        1 S>R ( POS )
                    #23 ZWO @B =
                    ja?   1 +             16
                    nein: #42 ZWO @B =
                          ja?    1 +       2
                          nein:           10
               dann dann  dann   BASIS !
 VERT je BASIS @ * I @B DUP #3A < ja? #30 -
                                  dann
                         DUP #40 > ja? #37 -
                                  dann
         DUP BASIS @ >= ZWO <0 ODER ja? ( FEHLER ) WEG 0 RDU
                                    dann +
      nun R>S * VERT ;n


:n COMPILER  $NAME
ja? $SUCH
    1   ( FUER WEITER ) BEM @B
        ja? ZWO 'n RUMPELSTILZCHEN
                = ja?   ( RUMP. ) 0 BEM !
                  nein: ( NICHT RUMP. ) Z-LESEN @
                        ja?   PWEG 0 1
                        nein: ZWO BEA @ <
                              ja? IE WEG 0
                              dann
                        dann
                  dann
        dann
   ja?  ( WEITERFLAG ? )  DUP =0
        ja? ( NUMBERPROCESSOR )
         WEG $ZAHL

          ja?  COMPILEFLAG @B
            ja?
               'n 2BLITERAL $DEP $DEP
            nein: BEM @B ja? EINGABEZAHL ! 0 Z-LESEN !
                         dann
            dann
          nein: IE
          dann
      nein: ( FOUNDPROCESSOR ) DUP 6 - @B #C0 UND
            COMPILEFLAG @B ODER
```

```
            DUP 1 =
            ja?   WEG HIER $ML >=U ja?  WEG MEMMESSAGE SYSWRITE
                                  nein: $DEP
                                  dann
            nein: DUP #80 = VERT #C1 = ODER
                  ja?    IE
                  nein: R>S $V1 ! $TUE $V1 @ S>R
                  dann
            dann
      dann

( this part has been changed to accomodate the Am1601 PSC/PSP  )
( way of doing things. To avoid an Am1601 exception the stack  )
( is loaded with 4 junk items on Reset. $SL is initialized to  )
( #0004. Stack underflow is indicated by $PSHOLEN returning a  )
( value below this                                             )

      $PSHOLEN $SL < ja? $SL $PSSETZEN STACKMESSAGE SYSWRITE
                       WEG $F1
                  dann
   dann
dann READYFLAG @B $P2 @B UND
   ja?  #20 TV8 !B
        TV8 DUP 1 + $PI @ TV8 - 1 - L>>>

  TV8 $CEN
   dann  ;n

(                  Compiler Auxiliary routines                 )
(                  ---------------------------                 )

:n ENTRYSETUP
       HIER #0001 UND <>0 ja?
           HIER 1 + $h !
       dann
       $F1 $KK ! $NAME DUP
       ja?   $SUCH =0 NICHT $RS @ UND
             ja?   RSMESSAGE SYSWRITE WEG  0
             nein: HIER DUP $KK ! LINK @ H2INC H2INC
                   $DEP $ND ZWO 4 >>> LINK ! HIER VERT H2INC
             dann
       nein: NAMEMESSAGE SYSWRITE
       dann ;n

:n $GETADR $NAME ja?   $SUCH DUP =0
                             ja?   IE      0
                             nein:         1
                             dann
                nein: NAMEMESSAGE SYSWRITE 0
                dann ;n
:hpri  '     $GETADR ja? COMPILEFLAG @
                       ja? 'n 2BLITERAL $DEP $DEP
```

```
                        dann
                    dann ;n

:prior  ;     'n RETEX $DEP  0 COMPILEFLAG !B
         $F2 <>
         ja? STRUCMESSAGE $LANG  + SYSLINE #20 + 16 >>>
             LINK @ DUP $H ! 4 + @ LINK !      0 IE
         dann ;n

:int  :   ENTRYSETUP ja? DEFEX VERT ! 1 COMPILEFLAG !B $F2
                    dann ;n
:n PRIMODIFY   $KK @ @B ODER $KK @ !B ;n

:int :PRIOR  i> 'n  : $dep <i      #80 PRIMODIFY ;n
:int :HPRI   i> 'n  : $dep <i      #40 PRIMODIFY ;n
:int :INT    i> 'n  : $dep <i      #C0 PRIMODIFY ;n

:prior JA? 'n BRONZ $DEP HIER H2INC ;n
:prior DANN  HIER VERT ! ;n
:prior NEIN: 'n JUMP $DEP HIER H2INC VERT i> 'n DANN $dep <i ;n
:prior JE    'n $JEEX $DEP HIER H2INC ;n
:prior NUN   'n LOOPEX  $DEP DUP i> 'n DANN $dep <i 2 + $DEP ;n
:prior +NUN  'n +LOOPEX $DEP DUP i> 'n DANN $dep <i 2 + $DEP ;n
:prior ANFANG HIER ;n
:prior ENDE? 'n BRONZ $DEP $DEP ;n
:prior DANN/NOCHMAL VERT 'n JUMP $DEP $DEP i> 'n DANN $dep <i ;n

:int KON        ENTRYSETUP ja? CONSEX VERT ! $DEP
                           dann ;n
:int VAR        ENTRYSETUP ja? VAREX  VERT ! $DEP
                           dann ;n
:int FELD       ENTRYSETUP ja? VAREX  VERT ! HIER + $H !
                           dann ;n

   'n TV4 02 +n $OC @n var SP               (  Screen Pointer )
:n !CHAR  SP @ !B SP INCR ;n
:n TLITERAL  I 1 + R>S @B PDUP + S>R SP @ PDUP + SP !
           VERT >>> ;n
:hpri " $PI INCR $PI @ 0 ZWO DUP 257 + DUP $TVE >
                                      ja? WEG $TVE dann
        je $PI @ @B #22 =
           ja?
           R>S PWEG 1 I S>R
           dann  $PI INCR
        nun
        ZWO $PI @ 2 - VERT - DUP >0 RDO UND
        ja?  COMPILEFLAG @
             ja?
               S>R I  'n TLITERAL $DEP HIER !B
               $H INCR HIER I >>> HIER  R>S + $H !
             dann
        nein:   TEXTMESSAGE SYSWRITE VERT WEG
```

```
        dann   ;n

:int !T  VERT  >>>  ;n

:n LEERZ  S>R SP @ #20 ZWO !B DUP 1 + R>S 1 - L>>> ;n

:int OK   SP @ SYSLINE SP !  #40 LEERZ  SP !  ;n

:n !FK    S>R I 2 * + 1 R>S je 2 - DUP S>R ! R>S
                           nun WEG ;n

:n WAND BASIS @ 10 = ja?   DUP ( ZAHL ) <0
                          ja? CHS #2D ( - ) !CHAR
                          dann 10000 0 ( W.-ANFANG )
                    nein: 16 BASIS ! #23 ( # ) !CHAR
                          #1000 1 ( W.-ANFANG )
                    dann  S>R
   anfang VERT ZWO /MOD VERT
             I NICHT ja?  DUP >0 ja? R>S WEG 1 S>R
                    dann        dann
                 I ja?  DUP  #30 + DUP #39 >
                                   ja? 7 +
                                   dann !CHAR
                    dann
                    WEG VERT BASIS @ / DUP =0
   ende? PWEG R>S NICHT ja? #30 !CHAR
                        dann ;n


:n $INSERT  VERT  #7  UND    2  *  KETTE  +  !  ;n

:n $CHAINACT  COMPILEFLAG  @
              ja?    'n  2BLITERAL  $DEP  $DEP
                     'n  $INSERT    $DEP
              nein:  ZWO  #FFF8  UND           ( mask for 0-7 )
                     =0  ja?    $INSERT
                         nein:  IE
                         dann
              dann  ;n

:hpri AUSH   'n  RUMPELSTILZCHEN  $CHAINACT  ;n
:hpri EINH   $GETADR  ja?  $CHAINACT
                      dann ;n

32 feld STACKBUF      ( Temporary Storage for Parameter Stack )
#0 var SCOUNT

:n ZEIG-STAPEL
   $P2 @ ja?                          ( End of block reached? )
       SP @
       S>R                          ( Save SP on return stack )
       $TVS SP ! #80 LEERZ              ( Blank Stack Display )
```

```
       $PSHOLEN $SL -                          ( Get # entries )
       DUP 16 > ja?
          WEG 16                          ( Max of 16 to display )
       dann
       DUP >0 ja?
            #1 - DUP SCOUNT !
            00 VERT je                ( copy p-stack to memory )
                I DUP + STACKBUF + !
            nun
            SCOUNT @ #1 + S>R
            anfang R>S DUP >0 ja?
               1 - DUP S>R
               DUP + STACKBUF + @
               DUP $TVS SCOUNT @ I - 8 * + SP ! WAND
            dann/nochmal
            WEG
        nein:
           WEG                                  ( discard count )
        dann
        R>S SP !                                   ( restore SP )
   dann
;n

:int ?  $GETADR  ja? 2 +
                 dann  ;n

:n SCHREIB    S>R SP @ I >>> SP @ R>S + SP ! ;n

:int WEG/AB
   $GETADR  ja?  DUP  $LL  VERT  >=U
                 ja?    IE
                 nein:  2  -  DUP  @  LINK  !  4  -  $H  !
                 dann
           dann  ;n

(                      End IPS-F1G                              )

(                Utilities and extensions                       )
(                ------------------------                       )

:int AWEG $SL $PSSETZEN ;n                      (  Clear Stack )

:n LANG <>0  ja?    L-STACKMESSAGE  STACKMESSAGE  -
             nein:  0
             dann  'n $LANG 2 + !  ;n

:n S-ON i> 'n 0           $dep  'n 2BLITERAL $dep
           'n ZEIG-STAPEL $dep  'n $INSERT   $dep <i ;n


(               Build the rest of SYSPAGE etc                   )
(               ----------------------------                    )
```

```
'n COMPILER                                  (  Construct chain )
'n ZEIG-STAPEL
'n RUMPELSTILZCHEN dup dup dup dup dup dup
'n JUMP #0500 #0500 $OC 11 !fk

             hier 'n $LL #2 +n !O                  ( Set $LL )
      hier 'n $H  #2 +n $OC @n !O                   ( Set $H )
     $$tv8 'n $PI #2 +n $OC @n !O             ( Initialise $PI )
           'n $ND #2 +n  #053E !O         ( Pointer for $SCODE )
$ccodes 22 +n @n 'n LINK #2 +n !O                  ( Set LINK )

               $$tv8 $$kbdip $OC !n             ( INPUTPOINTER )
               $$tve $$pe $OC !n
               #0 $$inson $OC !b              ( Insert Flag ON )
               #A0 $$tv8 $OC !b          ( Initial Blob Cursor )
               #0 $$readyflag $OC !b

               ( End of metacompilation )


(         Save IPS-F1G binary image; compilation off          )
(         ------------------------------------------          )

#0000 $OC hier $OC ~ IPS-F1G.BIN ~  $save  <X
~    IPS-F1G compiled OK    ~ #01D5 !t          ( info splash )
```

APPENDIX A – Machine Instruction Decoding Guide

1st BYTE		2nd BYTE	BYTES 3 & 4	Am1601 Instruction Format
HEX	BINARY			
00	0000 0000	<b><c>	-	#0<b><c> sJMP
01	0000 0001	<b><c>	-	#1<b><c> sJMP
02	0000 0010	<b><c>	-	#2<b><c> sJMP
03	0000 0011	<b><c>	-	#3<b><c> sJMP
04	0000 0100	<b><c>	-	#4<b><c> sJMP
05	0000 0101	<b><c>	-	#5<b><c> sJMP
06	0000 0110	<b><c>	-	#6<b><c> sJMP
07	0000 0111	<b><c>	-	#7<b><c> sJMP
08	0000 1000	<b><c>	-	#8<b><c> sJMP
09	0000 1001	<b><c>	-	#9<b><c> sJMP
0A	0000 1010	<b><c>	-	#A<b><c> sJMP
0B	0000 1011	<b><c>	-	#B<b><c> sJMP
0C	0000 1100	<b><c>	-	#C<b><c> sJMP
0D	0000 1101	<b><c>	-	#D<b><c> sJMP
0E	0000 1110	<b><c>	-	#E<b><c> sJMP
OF	0000 1111	<b><c>	-	#F<b><c> sJMP
10	0001 0000	<b><c>	-	#0<b><c> sJSR
11	0001 0001	<b><c>	-	#1<b><c> sJSR
12	0001 0010	<b><c>	-	#2<b><c> sJSR
13	0001 0011	<b><c>	-	#3<b><c> sJSR
14	0001 0100	<b><c>	-	#4<b><c> sJSR
15	0001 0101	<b><c>	-	#5<b><c> sJSR
16	0001 0110	<b><c>	-	#6<b><c> sJSR
17	0001 0111	<b><c>	-	#7<b><c> sJSR
18	0001 1000	<b><c>	-	#8<b><c> sJSR
19	0001 1001	<b><c>	-	#9<b><c> sJSR
1A	0001 1010	<b><c>	-	#A<b><c> sJSR
1B	0001 1011	<b><c>	-	#B<b><c> sJSR
1C	0001 1100	<b><c>	-	#C<b><c> sJSR
1D	0001 1101	<b><c>	-	#D<b><c> sJSR
1E	0001 1110	<b><c>	-	#E<b><c> sJSR
1F	0001 1111	<b><c>	-	#F<b><c> sJSR
20	0010 0000	<b><c>	-	#0<b><c> sLOAD
21	0010 0001	<b><c>	-	#1<b><c> sLOAD
22	0010 0010	<b><c>	-	#2<b><c> sLOAD
23	0010 0011	<b><c>	-	#3<b><c> sLOAD
24	0010 0100	<b><c>	-	#4<b><c> sLOAD
25	0010 0101	<b><c>	-	#5<b><c> sLOAD
26	0010 0110	<b><c>	-	#6<b><c> sLOAD
27	0010 0111	<b><c>	-	#7<b><c> sLOAD
28	0010 1000	<b><c>	-	#8<b><c> sLOAD
29	0010 1001	<b><c>	-	#9<b><c> sLOAD

2A	0010 1010	<b><c>	-	#A<b><c> sLOAD
2B	0010 1011	<b><c>	-	#B<b><c> sLOAD
2C	0010 1100	<b><c>	-	#C<b><c> sLOAD
2D	0010 1101	<b><c>	-	#D<b><c> sLOAD
2E	0010 1110	<b><c>	-	#E<b><c> sLOAD
2F	0010 1111	<b><c>	-	#F<b><c> sLOAD
30	0011 0000	<b><c>	-	#0<b><c> sSTORE
31	0011 0001	<b><c>	-	#1<b><c> sSTORE
32	0011 0010	<b><c>	-	#2<b><c> sSTORE
33	0011 0011	<b><c>	-	#3<b><c> sSTORE
34	0011 0100	<b><c>	-	#4<b><c> sSTORE
35	0011 0101	<b><c>	-	#5<b><c> sSTORE
36	0011 0110	<b><c>	-	#6<b><c> sSTORE
37	0011 0111	<b><c>	-	#7<b><c> sSTORE
38	0011 1000	<b><c>	-	#8<b><c> sSTORE
39	0011 1001	<b><c>	-	#9<b><c> sSTORE
3A	0011 1010	<b><c>	-	#A<b><c> sSTORE
3B	0011 1011	<b><c>	-	#B<b><c> sSTORE
3C	0011 1100	<b><c>	-	#C<b><c> sSTORE
3D	0011 1101	<b><c>	-	#D<b><c> sSTORE
3E	0011 1110	<b><c>	-	#E<b><c> sSTORE
3F	0011 1111	<b><c>	-	#F<b><c> sSTORE
40	0100 0000	<f><g>	-	#0<f><g> sBR
41	0100 0001	<f><g>	-	#1<f><g> sBR
42	0100 0010	<f><g>	-	#2<f><g> sBR
43	0100 0011	<f><g>	-	#3<f><g> sBR
44	0100 0100	<f><g>	-	#4<f><g> sBR
45	0100 0101	<f><g>	-	#5<f><g> sBR
46	0100 0110	<f><g>	-	#6<f><g> sBR
47	0100 0111	<f><g>	-	#7<f><g> sBR
48	0100 1000	<f><g>	-	#8<f><g> sBR
49	0100 1001	<f><g>	-	#9<f><g> sBR
4A	0100 1010	<f><g>	-	#A<f><g> sBR
4B	0100 1011	<f><g>	-	#B<f><g> sBR
4C	0100 1100	<f><g>	-	#C<f><g> sBR
4D	0100 1101	<f><g>	-	#D<f><g> sBR
4E	0100 1110	<f><g>	-	#E<f><g> sBR
4F	0100 1111	<f><g>	-	#F<f><g> sBR
50	0101 0000	<f><g>	-	#0<f><g> sBSR
51	0101 0001	<f><g>	-	#1<f><g> sBSR
52	0101 0010	<f><g>	-	#2<f><g> sBSR
53	0101 0011	<f><g>	-	#3<f><g> sBSR
54	0101 0100	<f><g>	-	#4<f><g> sBSR
55	0101 0101	<f><g>	-	#5<f><g> sBSR
56	0101 0110	<f><g>	-	#6<f><g> sBSR
57	0101 0111	<f><g>	-	#7<f><g> sBSR
58	0101 1000	<f><g>	-	#8<f><g> sBSR
59	0101 1001	<f><g>	-	#9<f><g> sBSR
5A	0101 1010	<f><g>	-	#A<f><g> sBSR

5B	0101 1011	<f><g>	-	#B<f><g> sBSR
5C	0101 1100	<f><g>	-	#C<f><g> sBSR
5D	0101 1101	<f><g>	-	#D<f><g> sBSR
5E	0101 1110	<f><g>	-	#E<f><g> sBSR
5F	0101 1111	<f><g>	-	#F<f><g> sBSR
60	0110 0000	0<cc>	nnnn (LSB first)	nnnn cc cNLOAD
61	0110 0001	-		* RESERVED *
62	0110 0010	-		* RESERVED *
63	0110 0011	-		* RESERVED *
64	0110 0100	-		* RESERVED *
65	0110 0101	-		* RESERVED *
66	0110 0110	-		* RESERVED *
67	0110 0111	-		* RESERVED *
68	0110 1000	-		* RESERVED *
69	0110 1001	-		* RESERVED *
6A	0110 1010	-		* RESERVED *
6B	0110 1011	-		* RESERVED *
6C	0110 1100	-		* RESERVED *
6D	0110 1101	-		* RESERVED *
6E	0110 1110	-		* RESERVED *
6F	0110 1111	-		* RESERVED *
70	0111 0000	Unsigned value	-	uN uNLOAD
71	0111 0001	Signed value	-	sN sNLOAD
72	0111 0010	-	-	* RESERVED *
73	0111 0011	-	-	* RESERVED *
74	0111 0100	-	-	* RESERVED *
75	0111 0101	-	-	* RESERVED *
76	0111 0110	-	-	* RESERVED *
77	0111 0111	-	-	* RESERVED *
78	0111 1000	-	-	* RESERVED *
79	0111 1001	-	-	* RESERVED *
7A	0111 1010	-	-	* RESERVED *
7B	0111 1011	-	-	* RESERVED *
7C	0111 1100	-	-	* RESERVED *
7D	0111 1101	-	-	* RESERVED *
7E	0111 1110	-	-	* RESERVED *
7F	0111 1111	-	-	* RESERVED *
80	1000 0000	0<cc>	nnnn (LSB first)	nnnn cc cJSR
81	1000 0001	0<cc>	nnnn (LSB first)	nnnn cc cJMP
82	1000 0010	0<cc>	nnnn (LSB first)	nnnn cc cLOAD
83	1000 0011	0<cc>	nnnn (LSB first)	nnnn cc cSTORE
84	1000 0100	0<cc>	nnnn (LSB first)	nnnn cc cLOADB
85	1000 0101	0<cc>	nnnn (LSB first)	nnnn cc cSTOREB
86	1000 0110	-	-	* RESERVED *
87	1000 0111	-	-	* RESERVED *
88	1000 1000	0<cc>	-	cc cpJSR
89	1000 1001	0<cc>	-	cc cpJMP
8A	1000 1010	0<cc>	-	cc cpLOAD
8B	1000 1011	0<cc>	-	cc cpSTORE

8C	1000 1100	0<cc>	-	cc cpLOADB
8D	1000 1101	0<cc>	-	cc cpSTOREB
8E	1000 1110	0<cc>	-	cc cRTS
8F	1000 1111	0<cc>	-	cc cpBSR
90	1001 0000	Signed value	-	ee EQ cBR
91	1001 0001	Signed value	-	ee NE cBR
92	1001 0010	Signed value	-	ee CS cBR
93	1001 0011	Signed value	-	ee CC cBR
94	1001 0100	Signed value	-	ee MI cBR
95	1001 0101	Signed value	-	ee PL cBR
96	1001 0110	Signed value	-	ee VS cBR
97	1001 0111	Signed value	-	ee VC cBR
98	1001 1000	Signed value	-	ee HS cBR
99	1001 1001	Signed value	-	ee LO cBR
9A	1001 1010	Signed value	-	ee GE cBR
9B	1001 1011	Signed value	-	ee LT cBR
9C	1001 1100	Signed value	-	ee GT cBR
9D	1001 1101	Signed value	-	ee LE cBR
9E	1001 1110	Signed value	-	ee AL cBR
9F	1001 1111	Signed value	-	ee NEF cBR
A0	1010 0000	Unsigned value	-	uN uN ADD
A1	1010 0001	Unsigned value	-	uN uN ADC
A2	1010 0010	Unsigned value	-	uN uN SUB
A3	1010 0011	Unsigned value	-	uN uN SBC
A4	1010 0100	Unsigned value	-	uN uN RSUB
A5	1010 0101	Unsigned value	-	uN uN RSBC
A6	1010 0110	Unsigned value	-	uN uN AND
A7	1010 0111	Unsigned value	-	uN uN OR
A8	1010 1000	Unsigned value	-	uN uN EOR
A9	1010 1001	-	-	* RESERVED *
AA	1010 1010	Unsigned value	-	uN uN RCMP
AB	1010 1011	Unsigned value	-	uN uN CMP
AC	1010 1100	Unsigned value Lower Nibble Only	-	uN uN MASK
AD	1010 1101	-	-	* RESERVED *
AE	1010 1110	Unsigned value	-	uN uN TST
AF	1010 1111	-	-	* RESERVED *
B0	1011 0000	Signed value	-	sN sN ADD
B1	1011 0001	Signed value	-	sN sN ADC
B2	1011 0010	Signed value	-	sN sN SUB
B3	1011 0011	Signed value	-	sN sN SBC
B4	1011 0100	Signed value	-	sN sN RSUB
B5	1011 0101	Signed value	-	sN sN RSBC
B6	1011 0110	Signed value	-	sN sN AND
B7	1011 0111	Signed value	-	sN sN OR
B8	1011 1000	Signed value	-	sN sN EOR
B9	1011 1001	-	-	* RESERVED *
BA	1011 1010	Signed value	-	sN sN RCMP
BB	1011 1011	Signed value	-	sN sN CMP

BC	1011 1100	-	-	* RESERVED *
BD	1011 1101	-	-	* RESERVED *
BE	1011 1110	Signed value	-	sN sN TST
BF	1011 1111	-	-	* RESERVED *
C0	1100 0000	See Appendix B	-	ALUP1
C1	1100 0001	See Appendix B	-	STACK
C2	1100 0010	See Appendix B	-	SHIFT
C3	1100 0011	-	-	* RESERVED *
C4	1100 0100	-	-	* RESERVED *
C5	1100 0101	-	-	* RESERVED *
C6	1100 0110	-	-	* RESERVED *
C7	1100 0111	-	-	* RESERVED *
C8	1100 1000	-	-	* RESERVED *
C9	1100 1001	-	-	* RESERVED *
CA	1100 1010	-	-	* RESERVED *
CB	1100 1011	-	-	* RESERVED *
CC	1100 1100	-	-	* RESERVED *
CD	1100 1101	-	-	* RESERVED *
CE	1100 1110	-	-	* RESERVED *
CF	1100 1111	-	-	* RESERVED *
D0	1101 0000	#<qq>0	-	qq PUSHPS
D1	1101 0001	#<qq>0	-	qq POPPS
D2	1101 0010	#<qq>0	-	qq PUSHRS
D3	1101 0011	#<qq>0	-	qq POPRS
D4	1101 0100	#<m>0	-	m SET
D5	1101 0101	#<m>0	-	m CLEAR
D6	1101 0110	-	-	* RESERVED *
D7	1101 0111	-	-	* RESERVED *
D8	1101 1000	-	-	* RESERVED *
D9	1101 1001	-	-	* RESERVED *
DA	1101 1010	-	-	* RESERVED *
DB	1101 1011	-	-	* RESERVED *
DC	1101 1100	-	-	*RESERVED*
DD	1101 1101	-	-	* RESERVED *
DE	1101 1110	-	-	* RESERVED *
DF	1101 1111	-	-	* RESERVED *
E0	1110 0000	-	-	* RESERVED *
E1	1110 0001	-	-	* RESERVED *
E2	1110 0010	0<cc>	nnnn (LSB first)	nnnn cc cIN
E3	1110 0011	0<cc>	nnnn (LSB first)	nnnn cc cOUT
E4	1110 0100	0<cc>	nnnn (LSB first)	nnnn cc cINB
E5	1110 0101	0<cc>	nnnn (LSB first)	nnnn cc cOUTB
E6	1110 0110	-	-	* RESERVED *
E7	1110 0111	-	-	* RESERVED *
E8	1110 1000	-	-	* RESERVED *
E9	1110 1001	-	-	* RESERVED *
EA	1110 1010	0<cc>	-	cc cpIN
EB	1110 1011	0<cc>	-	cc cpOUT
EC	1110 1100	0<cc>	-	cc cpINB

ED	1110 1101	0<cc>	-	cc cpOUTB
EE	1110 1110	-	-	* RESERVED *
EF	1110 1111	-	-	* RESERVED *
F0	1111 0000	0<cc>	-	cc EMULATE
F1	1111 0001	0<cc>	-	cc EXECUTE
F2	1111 0010	0<cc>	-	cc PREPARE
F3	1111 0011	-	-	* RESERVED *
F4	1111 0100	-	-	* RESERVED *
F5	1111 0101	-	-	* RESERVED *
F6	1111 0110	0<cc>	-	cc REFRESH
F7	1111 0111	-	-	* RESERVED *
F8	1111 1000	#00	-	DFX
F9	1111 1001	-	-	* RESERVED *
FA	1111 1010	-	-	* RESERVED *
FB	1111 1011	#00	-	2BLIT
FC	1111 1100	#00	-	JPPC
FD	1111 1101	#00	-	XB
FE	1111 1110	-	-	* RESERVED *
FF	1111 1111	0<cc>	-	cc FLAG

APPENDIX B – Machine Instruction Encoding Matrix

	0	1	2	3	4	5	6	7
0	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	cNLOAD	uNLOAD
1	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	sNLOAD
2	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
3	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
4	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
5	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
6	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
7	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
8	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
9	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
A	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
B	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
C	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
D	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
E	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*
F	sJMP	sJSR	sLOAD	sSTORE	sBR	sBSR	*	*

* Reserved

	8	9	A	B	C	D	E	F
0	cJSR	EQ cBR	uN ADD	sN ADD	ALUP1	PUSHPS	*	EMULATE
1	cJMP	NE cBR	uN ADC	sN ADC	STACK	POPPS	*	EXECUTE
2	cLOAD	CS cBR	uN SUB	sN SUB	SHIFT	PUSHRS	cIN	PREPARE
3	cSTORE	CC cBR	uN SBC	sN SBC	*	POPRS	cOUT	*
4	cLOADB	MI cBR	uN RSUB	sN RSUB	*	SET	cINB	*
5	cSTOREB	PL cBR	uN RSBC	sN RSBC	*	CLEAR	cOUTB	*
6	*	VS cBR	uN AND	sN AND	*	*	*	REFRESH
7	*	VC cBR	uN OR	sN OR	*	*	*	*
8	cpJSR	HS cBR	uN EOR	sN EOR	*	*	*	DFX
9	cpJMP	LO cBR	*	*	*	*	*	*
A	cpLOAD	GE cBR	un RCMP	sN RCMP	*	*	cpIN	*
B	cpSTORE	LT cBR	uN CMP	sN CMP	*	*	cpOUT	2BLIT
C	cpLOADB	GT cBR	uN MASK	*	*	*	cpINB	JPPC
D	cpSTOREB	LE cBR	*	*	*	*	cpOUTB	XB
E	cRTS	AL cBR	uN TST	sN TST	*	*	*	*
F	cpBSR	NEF cBR	*	*	*	*	*	FLAG

* Reserved

ALUP1 GROUP (#C0)

2nd Byte	Instruction
#00	P1 ADD
#10	P1 ADC
#20	P1 RSUB
#30	P1 RSBC
#40	P0 SUB
#50	P0 SBC
#60	P1 AND
#70	P1 OR
#80	P1 EOR
#90	NOP
#A0	P0 CMP
#B0	P1 RCMP
#C0	P0 MASK
#D0	CPL
#E0	P1 TST
#F0	NEG

STACK GROUP (#C1)

2nd Byte	Instruction
#00	DUPL
#10	DEL
#20	SWAP
#30	SOT
#40	RTU
#50	RTD
#60	PTOR
#70	RTOP
#80	IDX
#90	XRP

SHIFT GROUP (#C2)

2nd Byte	Instruction
#00	LSL
#10	LSR
#20	ROL
#30	ROR
#90	ASR

APPENDIX C – GNU Free Documentation License

GNU Free Documentation License
Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others. This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software. We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language. A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them. The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text

editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”. Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only. The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3. You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects. If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages. If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one

year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public. It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles. You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard. You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one. The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or simply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice. The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work. In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects. You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this

License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

APPENDIX D – GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all. The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of

this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program. In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable. If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you

indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program. If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances. It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be

guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS