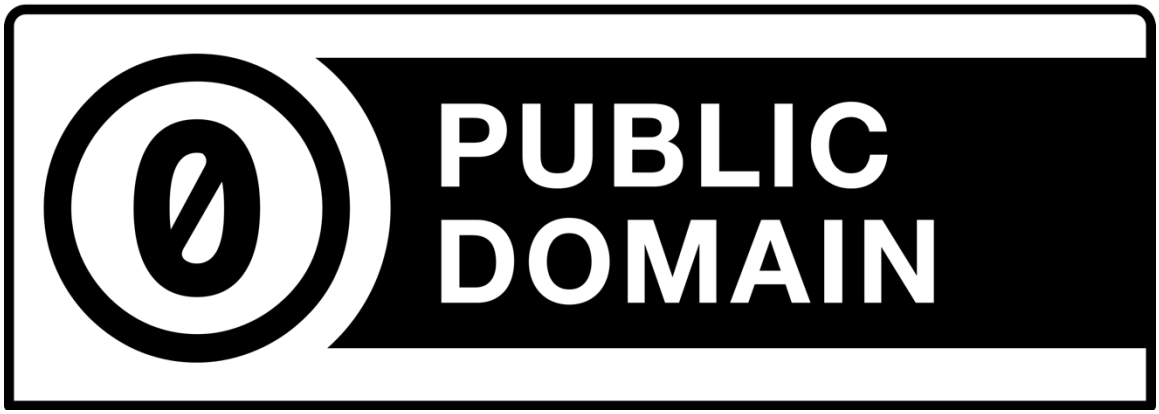
IPS Assemblers

Programmer's Reference

Issue 2.0.1
June 1, 2019

This document is subject to modification. Please contact the maintainer of the document for the latest version.

AMSAT-BDA



CC0 1.0 Universal (CC0 1.0)
Public Domain Dedication

No Copyright

- The person who associated a work with this deed has **dedicated** the work to the public domain by waiving all of his or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law.

  You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission. See **Other Information** below.

Other Information

- In no way are the patent or trademark rights of any person affected by CC0, nor are the rights that other persons may have in the work or in how the work is used, such as publicity or privacy rights.
- Unless expressly stated otherwise, the person who associated a work with this deed makes no warranties about the work, and disclaims liability for all uses of the work, to the fullest extent permitted by applicable law.
- When using or citing the work, you should not imply endorsement by the author or the affirmer.

The first version of this document was written
by Robin A. Gape, G8DQX

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	September, 1985	Initial version by Robin Gape (RG).
Revision	2.0.0	October 24, 2002	Converted from WordStar format to MS Word, and reformatted by Paul Willmott (PW)
Revision	2.0.1	June 1, 2019	Document placed in the public domain (PW)

IPS ASSEMBLERS

The following introduction to assembler programming in IPS explains the general principles involved. Briefly: DON'T. If you must, this introduction must be supplemented by the processor specific documentation for the processor that you will be using, and the machine specific documentation for the target that you will attempt to program on.

The principal reason for the development of the IPS assemblers is to allow the compilation of IPS for a particular target machine. (Normally this is a cross or X-Compilation)

Available assemblers

IPS assemblers have been written for the following processors:

- RCA 1802
- 6502
- 6800
- 6809
- 8080/Z80
- Am1601
- ARM

Use of assemblers in IPS

IPS has sufficient resources to program most problems without having to be intimately familiar with either the structure or the machine language of any particular target machine. The 3 major exceptions are:

i) to interface to particular hardware. The I/O structure of a particular IPS implementation may not be capable of the desired interface without modification, particularly if interrupts are involved.

ii) to program a time critical problem which can not be allowed any overhead. The IPS emulator has an overhead of typically 25 microseconds per executed word. Further, IPS treats all numbers as 16-bit. This results in an overhead if only bytes need to be manipulated. These two effects result in IPS programs typically running about 2 to 3 times slower than optimum machine code.

iii) to program special mathematical operations, such as rotational operations which would be inefficient using the standard IPS operators.

All these classes of problems are addressed by the ability to define new IPS words in terms of machine instructions rather than other IPS words. These newly defined words can then be used in the same fashion as other IPS words, but giving the programmer machine level access.

The IPS assembler provides the ability to define new words for a particular processor/machine. For the programmer to use an assembler successfully they must be familiar with the particular assembler that is used with their target machine.

Assembler applications

The IPS assemblers are intended mainly for short routines, that interface between high level IPS, the processor and application hardware. The IPS assemblers are not intended to support extended programs.

Assembly programs are hard to debug, often use more memory than the high-level equivalent, and are machine specific. Assembler routines cannot easily be transferred from one processor to another.

When programming time critical routines in assembler, it is usually only worthwhile assembly coding the inner loop.

Assemblers are for consenting adults (!) only

Assembler philosophy.

The assemblers have been kept as simple and unfussy as possible. Note that no special action is required if it is desired to assemble code for one sort of processor on a machine with a different processor.

Assembler definition

IPS allows the programmer to define new words as a sequence of assembler instructions. Note that assembly takes place in keyboard mode, compilation mode is not entered. (A colon definition would result in the compiler entering compilation mode.)

CODE

The introduction to an IPS named assembly routine is the word CODE followed by a name.

NEXT

The end of an assembler sequence (routine) is indicated by the word NEXT. This returns control to the emulator, and is the usual way of finishing an IPS assembly sequence.

Note that more than one exit from an assembly sequence may be provided by the multiple use of NEXT.

```
CODE a_routine (names assembly routine)

  (first action)
  …

  (last action)

NEXT (returns control to the emulator)
```

Fig 1: General layout of CODE definition

RCODE

The IPS assemblers allow the dubious practice of multiple entries to a routine. This is possible by marking an entry point with the word HIER in a CODE routine, and following this with the word RCODE followed by the name of the new entry point.

```
CODE a_routine

  (a_routine actions)
  …

HIER (deposits address on stack for RCODE following)

  (actions common to a_routine and another_routine)

NEXT

RCODE another_routine (picks up routine address from stack)
```

Fig 2: Use of RCODE construct.

Hacking

When it is necessary to write assembly routines that will be called from other assembly routines, the assembler can be used directly. This is necessary to produce replacement interrupt routines, for example.

To provide a named address, the following technique is often used:

```
0 FELD routine_name

  (start of subroutine)
  …

RET (return from subroutine)
```

Fig 3: Naming a directly-called assembler routine.

Code deposit

An assembler mnemonic, e.g. LDA, results in the corresponding machine instruction being deposited at the location that $H points to (HERE) and in $H being incremented to point to the next free position.

Operand order

The order of specifying addresses and operation follows the usual IPS principle of:

source, destination, operand.

Most conventional (algebraic) assemblers use a different order.

Mnemonics

The IPS assemblers often use the customary mnemonics for a particular processor. Some mnemonics have been changed. Note that there is little commonality between the various IPS assemblers. Equally, there is a lack of commonality in various matters between different processors.

Structured control flow

The IPS assemblers use structuring words rather than labels and gotos. There are (usually) no explicit jump/branch mnemonics.

The control structures provided are:

a) <condition> Y? N: TH

b) <condition> Y? TH

c) BEGIN <condition> END

d) BEGIN <condition> Y? TH/AGAIN

a) provides an equivalent to an IF (JA? NEIN: DANN) construct;

b) provides an equivalent to an IF statement without an else clause;

c) provides a controlled loop equivalent to the high level ANFANG ENDE? construct;

d) provides a controlled loop equivalent to the high level ANFANG JA? DANN/NOCHMAL construct.

The <condition> options allowed are specific to each assembler.

```
<condition> Y?
  (done if condition is true)
N:
  (done if condition is not true)
TH
(always done)
```

Fig 4: Use of Y? N: TH construct

```
 <condition> Y?
   (only done if condition is true)
TH
(always done)
```

Fig 5: Use of Y? TH construct

```
BEGIN
  (repeated actions)
<condition> END
(loop only exited if condition is true)
```

Fig 6: Use of BEGIN END construct

```
BEGIN
  (start of repeated actions)
  …
<condition> Y? (loop test and exit point)
  …
  (end of repeated actions)
TH/AGAIN
(loop only exited if action is false)
```

Fig 7: Use of BEGIN Y? TH/AGAIN construct

Handling of jump addresses

As with high level control constructs, the stack is used to manipulate jump addresses.

When the word Y? is met, the compiler deposits the appropriate jump code at HERE, increments $H and the value of $H is put on the stack. $H is then incremented to leave room for an unresolved jump address, which will be provided (resolved) later to the address specified on the stack.

When the TH is met later, the Y? jump address can now be resolved. HERE is the address that is to be jumped to. TH takes the address of the jump address off the stack, and stores HERE as the address for the jump associated with the Y?.

If a N: is met, a unconditional branch is assembled, again with an unresolved address field and with the address of the jump address left on the stack. The address left by the Y? is removed and the Y? is resolved to point to the position following the N: jump.

The word BEGIN is equivalent to HERE, and simply leaves the current address on the stack. This address is used by the END as the address of its conditional jump.

MYOPIA. It is possible to break the IPS structure rules by stack manipulation. Such deviant and devious practices are both unwise and liable to lose you friends.

TH/AGAIN

Early IPS assemblers did not support the use of TH/AGAIN. If not already available, define TH/AGAIN as follows:

```
: TH/AGAIN  ({loop start} {exit address field})
  VERT      ({exit address field} {loop start})
  NEVER END (return to loop start)
            ({exit address field})
  TH        (back fill exit address for Y?)
; (TH/AGAIN)
```

Fig 8: Definition of TH/AGAIN

This definition is processor independent, and relies on the handling of the jump addresses described above.

Use of regular IPS words

Because the assembler functions in interpretative mode any valid IPS words can be used in between assembly mnemonics, for instance to manipulate addresses.

The exit word NEXT can be used as often as necessary to create multiple exit points from a code word.

Assembler macros

A frequently required sequence of mnemonics etc. may be made a normal IPS definition. Each time the definition is invoked the sequence is run through again as if each mnemonic etc. had been entered individually. Such macros can contain low level structuring: Y? N: TH, BEGIN END. Because the high level IPS structure words are different from those used by the assemblers, it is possible to use conditional assembly within a macro.

IPS assembler interface

Mostly, the interface between CODE definitions and IPS is via the parameter stack. At assembler level stack operations are explicit rather than automatically managed.

6502

The stack is maintained by a pointer and appropriate machine-code instructions. If the processor already has stack-operation instructions these are usually used for the parameter stack, with the return stack simulated by a software pointer. In the case of the 6502 the machine stack is used for the return stack, with a software pointer for the parameter stack.

Before any assembly programmer is let loose at an IPS system, IPS has already taken certain machine resources into use. Particularly, certain registers and memory areas will be used for specific purposes.

The conventions should be followed, or unpredictable consequences are likely to follow.

This information can be found from

1. the relevant IPS assembler manual, and

2. the implementation notes for the particular IPS version in use.

Machine Resources

IPS requires various machine resources to run on a particular computer. Sometimes IPS' requirements can be met by existing operating system provision, often it is necessary to bypass the host's provided operating system.

Stack and memory layout

Clearly, there are two ways that can be chosen of representing a 16-bit number on a byte oriented machine:

i) Reversed address in which the least significant byte is held in the lowest address of a byte pair.

ii) High-to-low address in which the most significant byte is held in the lowest address of a byte pair.

This is one option too many for safety!

Stack layout

The stack(s) always grow from high memory to low memory. The stack normally uses the memory convention native to the processor in use.

Code and data memory layout

IPS compiled code and data grows from low memory towards high memory. IPS is designed to use the reversed address convention. In the case of those implementations which do not (6800) special conventions are necessary to allow for this.

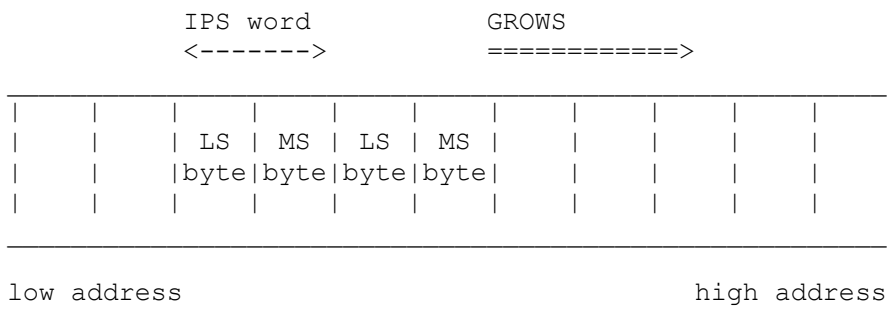
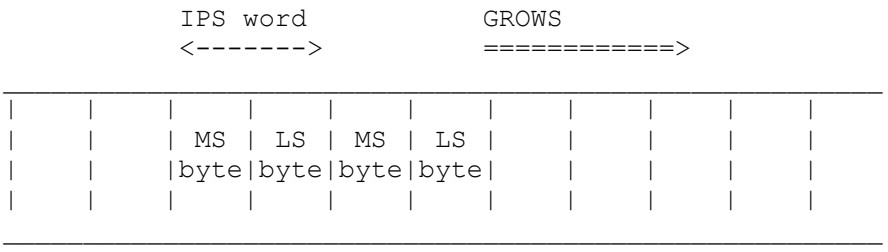


Fig 9: IPS code and data memory layout



N.B. high to low address convention.

Fig 10: IPS code and data memory layout (6800/68000)

Memory map

The memory maps of various IPS implementations vary, but tend to have a similar pattern, as typified below:

```
#FFFF     -|   Native operating system
           |
           |
           |   Hardware ports
#CFFF     -|   x    Start of parameter stack
           |   x
           |  \x/   parameter stack grows
           |
          -|
           |   |^    code grows
           |   |
           |   |
#5FFF     -|   ||   original IPS end
           |   ||
           |   ||   (compiled IPS code)
           |   ||
#400      -|   ||   start of IPS code
           |
           |        buffers etc.
           |
#200      -|
           |
           |        return stack (depth = 128 entries)
           |
#100      -|
           |
           |        emulator, etc
           |
#80       -|
           |
           |        operating system variables
           |
#00       -|
```

Fig 11: Typical IPS Memory Map

The memory map above is loosely based on the ATARI 6502 IPS implementation.

Note that the size of parameter stack is constrained by IPS code size.

APPENDIX A – Creative Commons CC0 1.0 Universal

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS DOCUMENT DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS PROVIDED HEREUNDER, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS PROVIDED HEREUNDER.

Statement of Purpose

The laws of most jurisdictions throughout the world automatically confer exclusive Copyright and Related Rights (defined below) upon the creator and subsequent owner(s) (each and all, an "owner") of an original work of authorship and/or a database (each, a "Work").

Certain owners wish to permanently relinquish those rights to a Work for the purpose of contributing to a commons of creative, cultural and scientific works ("Commons") that the public can reliably and without fear of later claims of infringement build upon, modify, incorporate in other works, reuse and redistribute as freely as possible in any form whatsoever and for any purposes, including without limitation commercial purposes. These owners may contribute to the Commons to promote the ideal of a free culture and the further production of creative, cultural and scientific works, or to gain reputation or greater distribution for their Work in part through the use and efforts of others.

For these and/or other purposes and motivations, and without any expectation of additional consideration or compensation, the person associating CC0 with a Work (the "Affirmer"), to the extent that he or she is an owner of Copyright and Related Rights in the Work, voluntarily elects to apply CC0 to the Work and publicly distribute the Work under its terms, with knowledge of his or her Copyright and Related Rights in the Work and the meaning and intended legal effect of CC0 on those rights.

1. Copyright and Related Rights. A Work made available under CC0 may be protected by copyright and related or neighboring rights ("Copyright and Related Rights"). Copyright and Related Rights include, but are not limited to, the following:

i. the right to reproduce, adapt, distribute, perform, display, communicate, and translate a Work;

ii. moral rights retained by the original author(s) and/or performer(s);

iii. publicity and privacy rights pertaining to a person's image or likeness depicted in a Work;

iv. rights protecting against unfair competition in regards to a Work, subject to the limitations in paragraph 4(a), below;

v. rights protecting the extraction, dissemination, use and reuse of data in a Work;

vi. database rights (such as those arising under Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, and under any national implementation thereof, including any amended or successor version of such directive); and

vii. other similar, equivalent or corresponding rights throughout the world based on applicable law or treaty, and any national implementations thereof.

2. Waiver. To the greatest extent permitted by, but not in contravention of, applicable law, Affirmer hereby overtly, fully, permanently, irrevocably and unconditionally waives, abandons, and surrenders all of Affirmer's Copyright and Related Rights and associated claims and causes of action, whether now known or unknown (including existing as well as future claims and causes of action), in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each member of the public at large and to the detriment of Affirmer's heirs and successors, fully intending that such Waiver shall not be subject to revocation, rescission, cancellation, termination, or any other legal or equitable action to disrupt the quiet enjoyment of the Work by the public as contemplated by Affirmer's express Statement of Purpose.

3. Public License Fallback. Should any part of the Waiver for any reason be judged legally invalid or ineffective under applicable law, then the Waiver shall be preserved to the maximum extent permitted taking into account Affirmer's express Statement of Purpose. In addition, to the extent the Waiver is so judged Affirmer hereby grants to each affected person a royalty-free, non transferable, non sublicensable, non exclusive, irrevocable and unconditional license to exercise Affirmer's Copyright and Related Rights in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "License"). The License shall be deemed effective as of the date CC0 was applied by Affirmer to the Work. Should any part of the License for any reason be judged legally invalid or ineffective under applicable law, such partial invalidity or ineffectiveness shall not invalidate the remainder of the License, and in such case Affirmer hereby affirms that he or she will not (i) exercise any of his or her remaining Copyright and Related Rights in the Work or (ii) assert any associated claims and causes of action with respect to the Work, in either case contrary to Affirmer's express Statement of Purpose.

4. Limitations and Disclaimers.

a. No trademark or patent rights held by Affirmer are waived, abandoned, surrendered, licensed or otherwise affected by this document.

b. Affirmer offers the Work as-is and makes no representations or warranties of any kind concerning the Work, express, implied, statutory or otherwise, including

without limitation warranties of title, merchantability, fitness for a particular purpose, non infringement, or the absence of latent or other defects, accuracy, or the present or absence of errors, whether or not discoverable, all to the greatest extent permissible under applicable law.

c. Affirmer disclaims responsibility for clearing rights of other persons that may apply to the Work or any use thereof, including without limitation any person's Copyright and Related Rights in the Work. Further, Affirmer disclaims responsibility for obtaining any necessary consents, permissions or other rights required for any use of the Work.

d. Affirmer understands and acknowledges that Creative Commons is not a party to this document and has no duty or obligation with respect to this CC0 or use of the Work.