

AMXTM Conversion Guide

for conversion of

AMX 86 v3

AMX 386 v1

AMX 68000 v2

applications for use with updated 32-bit versions of the

AMX Multitasking Executive

First Printing: June 1, 1999

Last Printing: November 1, 2002

Copyright © 1994 - 2002

KADAK Products Ltd.

206 - 1847 West Broadway Avenue

Vancouver, BC, Canada, V6J 1Y5

Phone: (604) 734-2796

Fax: (604) 734-8114

**Copyright © 1994-2002 by KADAK Products Ltd.
All rights reserved.**

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of KADAK Products Ltd., Vancouver, B.C., CANADA.

DISCLAIMER

KADAK Products Ltd. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability and fitness for any particular purpose. Further, KADAK Products Ltd. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of KADAK Products Ltd. to notify any person of such revision or changes.

TRADEMARKS

AMX in the stylized form and KwikNet are registered trademarks of KADAK Products Ltd. AMX, AMX/FS, InSight, *KwikLook* and *KwikPeg* are trademarks of KADAK Products Ltd. Microsoft, MS-DOS and Windows are registered trademarks of Microsoft Corporation. All other trademarked names are the property of their respective owners.

AMX CONVERSION GUIDE
Table of Contents

	Page
1. What's New	1
1.1 Introduction to AMX vc	1
1.2 AMX Comparisons	3
1.3 AMX vc New Features and Enhancements	4
2. Conversion Method	7
2.1 General Conversion Requirements	7
2.2 System Configuration Module	11
3. Application Code Conversion	13
3.1 Launch and Shutdown	13
3.2 Application Procedures Called by AMX	14
3.3 Application Calls to AMX Procedures	16
3.4 AMX 386 PC Supervisor Conversion	17

Appendices

Appendix A. AMX C Code Conversion	A-1
Appendix B. AMX Design Constants	B-1
B.1 AMX 386/ES Design Constants	B-1
B.2 AMX 386/EP Design Constants	B-2
B.3 AMX 68000 Design Constants	B-3
B.4 AMX CFire Design Constants	B-4
B.5 AMX PPC32 Design Constants	B-5
B.6 AMX 4-ARM, 4-Thumb Design Constants	B-6
B.7 AMX MA32 Design Constants	B-7
B.8 AMX 386/ET Design Constants	B-8

This page left blank intentionally.

1. What's New?

1.1 Introduction to AMX vc

AMX™ vc is the latest release of KADAK's AMX Multitasking Executive, the fifth in a long series spanning 20 years of development. AMX vc represents a conversion of KADAK's earliest assembler based AMX kernel to C. Changes to the internal functionality and application interface have resulted.

In an effort to ease the move of an existing AMX application to AMX vc, a conversion interface is provided. This interface makes the AMX vc kernel appear to the application to be the old familiar kernel. However, not all differences can be resolved by the conversion interface.

Differences between AMX vc and its predecessors are presented in this chapter. This summary will help identify AMX vc features which you may wish to avoid if you are planning to port a new AMX vc application to other target processors supported by previous AMX releases. The remainder of this document describes the changes that you will need to make to convert an existing AMX application to run under AMX vc.

AMX Product Nomenclature

Unless distinctions are necessary, the term **AMX xxx** will be used to refer to any of the following AMX implementations:

AMX 86 v3.0x	AMX for Intel 8088/8086, 80x88/80x86 (real mode)
AMX 386 v1.0x	AMX for Intel 80386, 80486 (protected mode)
AMX 68000 v2.0x	AMX for Motorola MC680x0, MC683xx

The term **AMX vc** will be used to refer to the following AMX implementations:

Product	Part Number	Target Processor(s)
AMX 386/ET	PN722-1	AMX for Intel 80386, 80486, Pentium (protected mode)
AMX 386/ES	PN812-1	AMX for Intel 80386, 80486, Pentium (protected mode) (for use with Beacon tools)
AMX 386/EP	PN814-1	AMX for Intel 80386, 80486, Pentium (protected mode) (includes PC BIOS and DOS support)
AMX 68000	PN532-1	AMX for Motorola MC680x0, MC683xx
AMX CFire	PN512-1	AMX for Motorola ColdFire® families
AMX PPC32	PN382-1	AMX for IBM and Motorola PowerPC® families
AMX 4-ARM	PN402-1	AMX for the ARM® families (ARM mode)
AMX 4-Thumb	PN422-1	AMX for the ARM® families (ARM + Thumb mode)
AMX MA32	PN442-1	AMX for MIPS32™ families

Filename Nomenclature

AMX filenames are of the following form.

<i>CJnnnFFF.XXX</i>	AMX filenames
<i>CJZZZ.H</i>	Generic AMX include file (copy of <i>CJnnn.H</i>)

The *nnn* in each AMX filename is the 3-digit string from the AMX part number used by KADAK to identify the AMX vc target processor. For example, file *CJnnnSD.H* is an AMX header file for any version of AMX vc. File *CJ532SD.H* is the same AMX header file for AMX 68000 which operates on the Motorola MC680x0 and MC683xx processors and is identified by KADAK part number PN532-1.

File *CJZZZ.H* is a generic include file which includes the subset of target specific AMX header files needed for compilation of your application C code. By including file *CJZZZ.H* in your source modules, your AMX application becomes readily portable to other target processors.

The generic include file *CJZZZ.H* is a copy of the corresponding part numbered AMX file. For example, if you are using AMX 68000, identified by KADAK part number PN532-1, the file *CJZZZ.H* is a copy of file *CJ532.H*.

1.2 AMX Comparisons

AMX uses a set of design constants which vary according to the constraints imposed by each target processor. These design constants are summarized below. Appendix B presents a target specific comparison table for each version of AMX vc.

Characteristic	AMX vc	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	see Appendix B	128	128	256
Minimum Interrupt Stack (bytes)	see Appendix B	128	256	256
Minimum task storage (bytes)	see Appendix B	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	see Appendix B	yes	yes	n/a
DOS Command Task	see Appendix B	yes	no	n/a
File Manager	see Appendix B	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	see Appendix B	little	little	big
Memory model	see Appendix B	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	see Appendix B	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

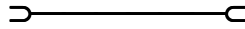
Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 supports the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

n/a: Not applicable

1.3 AMX vc New Features and Enhancements



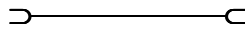
The **Buffer Pool Manager** has been enhanced to queue requests for buffers when no free buffers are available. Tasks can specify a maximum wait time and priority at which they are prepared to wait for a buffer to become available.

The AMX xxx ability to reinitialize any or all buffer pools was deemed to be unnecessary in all but the rarest of applications and is not allowed by AMX vc. The same effect can be achieved by deleting and recreating specific buffer pools.

AMX vc provides two other new features: the ability to derive a buffer pool id given only a buffer pointer and the ability to get the current status of any buffer pool.



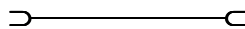
Task signals are no longer supported by AMX vc. Many of the capabilities of task signals overlapped with services provided by the Semaphore Manager and the Event Manager.



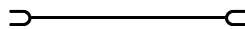
The **Event Manager** has added support for pulsed events. A pulsed event differs from a state driven event in that a pulsed event signal only stays true for the time that it takes AMX to wake any tasks waiting for the signal. With pulsed event signals, there is no need for an additional call to set the signals false. Any number of tasks can synchronize to a pulsed event group signal. AMX vc adds the ability to get the current status of an event group.



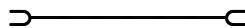
The **Message Exchange Manager** has been enhanced in a number of ways. AMX vc message exchanges support synchronous message passing. A task can send to an exchange and wait for acknowledgement from the receiver. An application can retrieve the current status of a message exchange and can flush all messages from a message exchange. The flush releases all queued messages and notifies all tasks waiting for acknowledgement that the flush occurred.

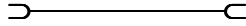


The **Mailbox Manager** is new for AMX vc. In essence, a mailbox is a message exchange with only one message queue. AMX vc mailboxes also support synchronous message passing. The Mailbox Manager is a lean alternative to the Message Exchange Manager when multiple message queues are not a requirement. Use message exchanges only when priority ordering of messages is needed.



A semaphore status call has been added to the **Semaphore Manager**.



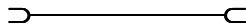


The **Memory Manager** has been revised to make its application interface more like that of other AMX vc managers. The Memory Definition Table and Memory Assignment Procedure have been eliminated. Pools of memory can be created or deleted dynamically in a manner similar to buffer pools. If you are familiar with AMX xxx handled memory, you will observe that AMX vc deals only with handled memory in which the handle is an AMX vc memory pool id.

Additional disjoint sections of memory can be dynamically added to existing memory pools. A private memory block can be carved from a large pool and used to create another memory pool. When the smaller memory pool is no longer required, it can be deleted and its storage can be released back to the larger memory pool from which it was derived.

Once a memory block has been allocated from a memory pool, the AMX vc Memory Manager gives you the ability to resize that block. You may release the unused upper portion thereby shrinking the memory block. You can also grow the block provided enough contiguous free memory exists immediately above the block.

The AMX vc Memory Manager includes a procedure to get the memory pool id for a previously allocated memory block.



All AMX vc tasks are trigger tasks. The task, once created, is started by triggering it with a call to AMX. The task can then wait on a mailbox or message exchange for messages if so desired.

However, for compatibility with AMX xxx and recognizing that many tasks do want to wait for messages, AMX vc offers a message exchange task. Tasks of this type have their own private message exchange from which AMX vc automatically retrieves and delivers messages to the task. Such tasks can be configured to receive messages on their stack by reference or by value.



This page left blank intentionally.

2. Conversion Method

2.1 General Conversion Requirements

Include Files

If you are programming in C, you will have to edit your C files to alter the list of included AMX header files to reflect the revised file names.

Replace your list of included AMX header files with two statements:

```
#include "CJZZZ.H"  
#include "CJnnnCV.H"
```

The file *CJZZZ.H* is the generic AMX vc header file which automatically includes the correct subset of new AMX vc header files. The file *CJnnnCV.H* is an AMX xxx to AMX vc conversion header which is provided with AMX vc. It handles procedure name and parameter changes, data type and constant definitions and procedure prototyping required to convert AMX xxx code references to the equivalent AMX vc references. File *CJnnnCV.H* must be included after the generic AMX vc header file *CJZZZ.H*.

C Code Conversion

Revise any C modules which reference AMX xxx to include the generic AMX vc header file and the AMX xxx to AMX vc conversion header file *CJnnnCV.H*. The conversion header handles most of the conversion. Changes to the names of constants, structures and procedures are accommodated by *#define* redefinitions. Parameter type changes and reordering are corrected using macro definitions with parameter casting where necessary.

Some AMX xxx features have no direct AMX vc counterpart. A conversion library *CJnnnCV.LIB* (or *CJnnnCV.A*) is provided with AMX vc to provide those features. Include this library when you link your AMX vc application.

You will also have to make some code changes to accommodate differences which cannot be handled by the conversion header or the conversion library. These changes are described in Chapter 3.

Assembler Code Conversion

AMX vc does not provide an explicit assembly language interface. To call an AMX vc procedure from assembly language, you must call the AMX vc C procedure adhering to the calling conventions dictated by the particular C compiler which you are using.

If you are converting from AMX xxx to AMX vc on a different target processor, you will have to recode your AMX xxx assembly language modules.

If you are converting from AMX xxx to AMX vc on the SAME target processor, then you can create an AMX xxx assembler to AMX vc C interface using the C interface modules provided with AMX xxx. Identify the subset of AMX assembler *AAxxxxxx* procedures which your application references. Extract the corresponding *ajxxxxxx* procedures from the AMX C interface modules (*ajxxxxxx.ASM*). Change the *AAxxxxxx* references in the extracted procedures to the equivalent *cjxxxxxx* procedures in AMX vc. Then change the extracted procedure names from *ajxxxxxx* to *AAxxxxxx*.

The result is an *AAxxxxxx* to *cjxxxxxx* conversion interface. Each of these conversion procedures must then be edited to take the AMX parameters from the registers in which they are received and move them to the registers or stack as required by the particular C compiler which you are using. Be sure to preserve all registers according to the AMX xxx assembly language interface definition for each procedure.

Error Codes

Most AMX error and warning codes are automatically translated from their AMX xxx *AERxxxx* form to the equivalent AMX vc *CJ_ERxxxx* or *CJ_WRxxxx* form by the AMX vc conversion header. In some cases, the translation is done by AMX xxx replacement procedures in the AMX vc conversion library.

The following AMX xxx error codes have been converted to warnings since they were never trapped to your AMX xxx User Error Procedure.

<i>AERCNW</i>	<i>CJ_WRAKNOWAIT</i>	see <i>ajwakc</i> , <i>ajwakcs</i>
<i>AERMNA</i>	<i>CJ_WRMMNOMEM</i>	see <i>ajmget</i> , <i>ajmgeh</i> , <i>ajmhan</i>
<i>AERNMG</i>	<i>CJ_WRMXEMPTY</i>	see <i>ajgmsg</i> , <i>ajmxget</i>
<i>AERSBY</i>	<i>CJ_WRSMINUSE</i>	see <i>ajsmget</i>
<i>AERTMO</i>	<i>CJ_WRTMOUT</i>	all procedures which time out
<i>AERTNW</i>	<i>CJ_WRTKWAKEN</i>	see <i>ajwake</i> , <i>ajsgnl</i>
<i>AERWBT</i>	<i>CJ_WRTKDELAY</i>	see <i>ajwatm</i>

The following AMX xxx error codes which were never trapped to your AMX xxx User Error Procedure have been converted to error codes.

<i>AERCWT</i>	<i>CJ_ERAKNEED</i>	see <i>ajgmsg</i>
<i>AERSIU</i>	<i>CJ_ERSMBUSY</i>	see <i>ajsmdel</i>

AMX xxx rejected timeout values < 0 with the error code *AERTMV*. AMX vc uses a timeout of < 0 to indicate that the caller does not want to wait if the requested service or resource is not immediately available. An AMX vc warning code is returned in such cases. The following AMX xxx procedures will now return a warning code if given a negative timeout value. Note that these warnings imply that you had an undetected fault in the AMX xxx system which you are converting.

<i>ajevwat</i>	<i>CJ_WREVNOEVT</i>
<i>ajmxwat</i>	<i>CJ_WRMXEMPTY</i>
<i>ajsmrsv</i>	<i>CJ_WRSMINUSE</i>
<i>ajsmwat</i>	<i>CJ_WRSMINUSE</i>

Following is a list of AMX xxx procedures which may return new AMX vc error codes not previously possible. Explanation of these error codes can be found in Appendix B of the AMX User's Guide for AMX vc.

<i>ajbcre</i>	<i>CJ_WRBMMEMSIZ</i>	
<i>ajbdel</i>	<i>CJ_ERBMBUSY</i>	
<i>ajbfre</i>	<i>CJ_ERNOENVLOP</i>	
<i>ajmxget</i>	<i>CJ_ERAKNEED</i> , <i>CJ_WRMXFLUSH</i>	
<i>ajmxwat</i>	<i>CJ_ERAKNEED</i> , <i>CJ_WRMXFLUSH</i>	
<i>ajsmfre</i>	<i>CJ_ERNOENVLOP</i>	
<i>ajsmrls</i>	<i>CJ_ERNOENVLOP</i>	
<i>ajsmsig</i>	<i>CJ_ERSMOVF</i>	
<i>ajtds</i>	<i>CJ_ERTMID</i> , <i>CJ_ERTMVALUE</i>	
<i>ajtkcre</i>	<i>CJ_ERTKSTORE</i>	
<i>ajtkdel</i>	<i>CJ_ERTKPR</i>	
<i>ajtktrm</i>	<i>CJ_ERTKTERM</i>	(<i>AERANA</i> is NOT returned)
<i>ajtmdel</i>	<i>CJ_ERTMBUSY</i>	

AMX Structures

Structure names and field names have changed in AMX vc. The conversion header file *CJnnnCV.H* minimizes the effects in a variety of ways.

AMX xxx structures which most applications frequently reference have been included in the AMX vc conversion header file without name changes. These structures include *amxtdts*, *amxsbs* and *amxbps*. Access to these structures and their fields remains the same as for AMX xxx. AMX vc procedures do not reference these structures.

The Task Control Block structure *amxtcbs* is still available. However, the offset of structure member *amtcbusr* has changed to reflect the new position of the 16 user bytes in the Task Control Block. In addition, the size of structure *amxtcbs* has changed.

The AMX time/date structure *amxtds* and its associated fields have undergone a name change only. The physical layout of the structure has not changed. In this case, the names of the structure and each of its fields have been translated (using *#define*) to their AMX vc names. Applications using *amxtds* or any of its members require no change.

The names of AMX xxx structures *amxufts*, *amxlhs*, *amxlns* and *amxlks* have been translated (using *#define*) to the equivalent AMX vc structure names. Since only AMX accesses these structures, the AMX xxx field names have not been redefined. Consequently, any reference to the fields in these structures using the AMX xxx field names will result in compilation errors as a reminder that your application is messing with private AMX information. Application code that accesses fields in any of the structures must be recoded to use the AMX vc field names.

AMX xxx structure *amxisps* is no longer defined. AMX vc Interrupt Service Procedures are now created as described in Chapter 3.2.

Configuration Modules

Revise your AMX System Configuration Module as described in Chapter 2.2.

AMX vc uses a Target Configuration Module to define processor specific parameters. This is a simple text file similar to your AMX xxx User Parameter File. The file must be created as described in Chapter 16 of the AMX User's Guide for AMX vc. The Target Configuration Module is then created as described in Chapter 4 of the target specific AMX vc Target Guide.

Linking

Compile your C modules, assemble your assembler modules, generate new object libraries (if you use libraries) and link your AMX application with the AMX vc object and library modules as described in the AMX vc Tool Guide for your toolset.

Be sure to include the AMX vc conversion library file *CJnnnCV.LIB* or (*CJnnnCV.A*) in your link. Include it immediately prior to the AMX vc Library.

2.2 System Configuration Module

Your System Configuration Module must be converted to meet AMX vc requirements. If you have a simple AMX xxx configuration, you will find that the quickest way to create your AMX vc System Configuration Module is to enter the parameters from scratch using the AMX vc Configuration Builder.

Alternatively, you can use the Configuration Generator to translate your AMX xxx User Parameter File *CFGXXX.UP* to an AMX vc User Parameter File *CFGVC.UP* as follows:

```
C:>CJnnnCG CFGXXX.UP CJnnnCV.CT CFGVC.UP
```

File *CJnnnCV.CT* is a conversion template file provided with AMX vc which is used by the AMX vc Configuration Generator (*CJnnnCG.EXE*) to convert your AMX xxx User Parameter File *CFGXXX.UP* to an AMX vc equivalent file *CFGVC.UP*.

You can then use the AMX vc Configuration Manager to read and modify file *CFGVC.UP* and generate a new System Configuration Module *CFGVC.C* as described in Chapter 15 of the AMX User's Guide for AMX vc.

If the AMX vc Configuration Manager rejects your new User Parameter File *CFGVC.UP*, it implies that one or more of your AMX xxx parameters are not valid for AMX vc use. The Configuration Manager will identify the errant parameter and/or indicate the line number in file *CFGVC.UP* which is at fault.

For example, if an AMX xxx task had a 128 byte stack, the AMX vc Configuration Manager will reject the task definition and insist that some increased minimum bytes of storage be provided. Use your text editor to correct the parameter in the task definition in file *CFGVC.UP* and try again.

One of the differences to note between AMX xxx and AMX vc is that AMX vc allocates the Task Control Block from the task stack storage provided. Developers must now account for this extra storage requirement when choosing each task's stack size.

Once you have a new System Configuration Module *CFGVC.C*, you must compile it and link it with your application as described in the AMX vc Tool Guide for the tools which you are using.

It is possible that you will encounter compilation errors when you first compile your new System Configuration Module *CFGVC.C*. If the C compiler indicates that line *n* of file *CFGVC.C* has an error, examine line *n* of the file. The error will probably have been intentionally generated using the `#error` preprocessing directive to draw your attention to the text at line *n* describing the reason for the conversion fault.

If you wish, you can use the new AMX vc Configuration Manager to generate a documentation text file describing your AMX vc configuration. Simply select *File* and *Document* from the Configuration Manager's menus.

If your AMX xxx message size was set >12 bytes, you must edit the definition of *CJ_MAXMSZ* in AMX vc header file *CJnnnAPP.H*. If your message size is >64 bytes (what are you doing?), you will also have to rebuild the AMX Conversion Library using the edited copy of file *CJnnnAPP.H*. Instructions for making the AMX vc libraries are provided in the toolset specific AMX Tool Guides.

Memory Assignment Procedure

The AMX xxx Memory Manager required that you provide a Memory Assignment Procedure, say *usermap*, to assign memory sections for its use. To port such an application to AMX vc, the following statement must be present in your AMX vc User Parameter File.

```
...MAP      usermap
```

If you used the Configuration Generator to convert your User Parameter File, the statement will have been generated for you. Otherwise, use your text editor to add the statement to the end of your AMX vc User Parameter File.

The keyword `...MAP` directs the AMX vc Configuration Generator to predefine an AMX xxx memory pool. At launch, AMX creates the pool and repetitively calls your C procedure *usermap* assigning each of your AMX xxx sections to the pool. All AMX xxx *ajmxxxx* calls to the Memory Manager are converted to reference this unique predefined memory pool.

3. Application Code Conversion

3.1. Launch and Shutdown

The AMX vc **Launch** and **Shutdown** are described in Chapter 2.4 of the AMX User's Guide for AMX vc.

The launch and shutdown are performed by procedures *ckslaunch* and *ckslleave* respectively. These procedures, coded in C, are located in AMX vc file *CJnnnUF.C* which is considered to be part of your application startup code. You can examine and, if necessary, modify and rebuild *CJnnnUF.C* to suit your needs. Include the compiled object module in your link specification.

3.2. Application Procedures Called by AMX

Error Procedures

The AMX vc **Fatal Exit** and **User Error Procedures** are described in Chapters 14.1 and 14.2 of the AMX User's Guide for AMX vc.

Fatal errors and user errors are funnelled through procedures *cjksfatal* and *cjkerror*. These procedures, coded in C, are located in file *CJnnnUF.C* which is considered to be part of your application startup code. You can examine and, if necessary, modify this module.

If you have an AMX xxx Fatal Exit Procedure, you may wish to modify procedure *cjksfatal* to call it.

If you have an AMX xxx User Error Procedure, you may wish to modify procedure *cjkerror* to call it.

In examining *CJnnnUF.C* you may notice the procedure *cjksbreak*. You can use your debugger to set a breakpoint on this procedure to retrieve valuable information if and when AMX vc ever detects a condition that should not occur in most working applications.

Interrupt Service Procedures

The rules for coding Interrupt Service Procedures have changed. Since AMX xxx Interrupt Supervisor procedures *ajint* and *ajinx* are not required by AMX vc, conforming Interrupt Service Procedures become simpler to code.

AMX vc Interrupt Service Procedures consist of an ISP root and an Interrupt Handler coded according to the rules defined in Chapter 4 of the AMX User's Guide for AMX vc. Examples are provided in the AMX Target Guide for each AMX vc target processor. The ISP root is automatically created for you by the AMX vc Configuration Generator from information provided in a new AMX Target Parameter File.

The Interrupt Handler can be derived from the body of your AMX xxx Interrupt Service Procedure. The Interrupt Handler is all of the code previously located between the *ajint* and *ajinx* calls. If you used AMX xxx procedure *ajispm* to make an AMX xxx C root ISP, your AMX vc Interrupt Service Procedure will be the same procedure which you provided in that call.

Restart and Exit Procedures

Timer Procedures

Trigger Task Procedures

The AMX vc calling sequence for these procedures matches that of AMX xxx.

Message Task Procedures

AMX xxx tasks which receive messages (have message mailboxes) are created as AMX vc message exchange tasks. Predefined tasks of this type are created automatically by AMX vc from information provided in your System Configuration Module. Your AMX xxx *ajtkcre* calls to dynamically create tasks of this type are forced by the AMX vc conversion library to build AMX vc message exchange tasks.

By default, AMX vc message exchange tasks receive messages passed by value. Since a message is a structure, the entire structure is passed to the task by value. You may find that your C compiler for the AMX vc target processor does not support such prototypes. If this is the case, you must force AMX vc to pass messages to your AMX xxx tasks by reference. Use the AMX vc Configuration Manager to edit your task definitions changing the message passing attribute from *value* to *reference* for each task that has message queues.

Your task procedures must then be prototyped as described in the *cjtkmxinit* procedure listing in Chapter 18 of the AMX User's Guide for AMX vc. Your task must use a pointer reference to access its message instead of explicit structure references.

Time/Date Scheduling Procedure

An AMX xxx Time/Date Scheduling Procedure received an AMX Time/Date structure on the stack. The structure was passed to the procedure by value.

An AMX vc Time/Date Scheduling Procedure receives a pointer to an AMX Time/Date structure stored on the stack. The structure is passed by reference, not by value.

Consequently a minor code change to your Time/Date Scheduling Procedure will be required.

Task Termination Procedure

An AMX vc Task Termination Procedure receives the task id of the task being terminated in addition to the reason for the termination.

Consequently a minor code change to your AMX xxx Task Termination Procedure prototype will be required to accommodate the new parameter, even though it can be ignored by your procedure.

Task Trap Handler User Scheduler Hooks

An AMX Task Trap Handler and all AMX Task Scheduler user hooks are target processor sensitive. Consequently, all of these procedures will have to be recoded to meet the requirements specified by AMX vc for the target processor of interest.

3.3 Application Calls to AMX Procedures

Most AMX xxx procedures operate in the AMX vc environment with no externally discernable differences. Calling sequence differences are hidden by the AMX vc header files and by the AMX vc conversion library.

In the tables which follow, the symbol □ next to an AMX xxx procedure name implies that the conversion is handled by recompilation of your modules using the AMX vc header file *CJnnnCV.H*.

The symbol ■ next to an AMX xxx procedure name implies that the conversion is handled by replacement procedures in the AMX vc conversion library. Modules containing references to these procedures must be recompiled using the AMX vc header file *CJnnnCV.H*.

AMX xxx procedures which have no □ or ■ next to them have no exact AMX vc equivalents because of changes introduced by AMX vc or because of target processor or C differences. If an alternate AMX vc procedure exists, its name is provided for your guidance. All code which references any of these procedures must be revised to reflect the requirements of your AMX vc target processor and/or the C tools which you use for it.

Interrupt State on Return

In most cases, the processor interrupt state upon return from an AMX vc procedure will mimic the operation of the corresponding AMX xxx procedure. If you have any doubt or specific concern, compare the AMX xxx and AMX vc descriptions of the particular procedures.

Two exceptions must be noted. AMX xxx procedures *ajsmwat* and *ajrmrsv* unconditionally returned with interrupts enabled. Their AMX vc counterparts now return with interrupts restored to the state they were in upon entry to the procedures. Therefore, if you called *ajsmwat* or *ajrmrsv* with interrupts disabled and expected AMX to automatically enable interrupts on return, you must add the code to explicitly enable the interrupts upon return.

3.4 AMX 386 PC Supervisor Conversion

General Conversion Process

The AMX 386/EP PC Supervisor is a direct adaptation of the AMX 386 v1 PC Supervisor. Unlike the AMX 386/EP kernel, the PC Supervisor is coded in assembly language permitting it to easily deal with the low level processor dependent features of the PC/AT, the ROM BIOS, MS-DOS[®] and the Phar Lap TNT DOS-Extender[™]. Porting the PC Supervisor in this manner ensures that AMX 386/EP benefits from the extensive testing and use that AMX 386 v1 has undergone.

The following AMX 386 v1 PC Supervisor procedures must be converted to their AMX 386/EP equivalents. You must include the AMX 386/EP header file *CJ814PCS.H* in your list of include files in any module which uses these procedures. The header file *CJ814PCS.H* provides the prototypes for the new AMX 386/EP procedures. It also provides a set of C macro definitions which, if required, will automatically convert AMX 386 v1 references to their AMX 386/EP counterparts.

AMX 386 v1	AMX 386/EP
<i>ajbeep</i>	<i>cjpspeaker</i>
<i>ajboot</i>	<i>cjpsboot</i>
<i>ajclop</i>	<i>cjpsclopoption</i>
<i>ajkbgc</i>	<i>cjpskbget</i>
<i>ajkbof</i>	<i>cjpskbmask</i>
<i>ajkbon</i>	<i>cjpskbmask</i>
<i>ajkbss</i>	<i>cjpskbhit</i>
<i>ajpdr1</i>	<i>cjpsr1s</i>
<i>ajpdrp</i>	<i>cjpsrsv</i>
<i>ajpdrs</i>	<i>cjpsrsv</i>
<i>ajspof</i>	<i>cjpspeaker</i>
<i>ajspou</i>	<i>cjpspeaker</i>

AMX 386/EP provides procedures *cjcf sint* and *cjcfregst* which ease the generation of ROM BIOS and DOS calls if such services are not offered by the C compiler. The register array *cjxregs* used by these procedures differs from the corresponding AMX 386 v1 structure *amxregs*.

As indicated in the conversion table in Chapter 3.3, the following AMX 386 v1 procedures are no longer provided by AMX 386/EP. These procedures duplicate features now offered by most C compilers.

ajproc, ajprocq
ajsint, ajsintq
ajmod1
ajgofs, ajgseg, ajgsreg
ajsofs, ajsseg, ajssreg

If you wish, you can port these AMX 386 v1 procedures by editing source modules *AJ881UA.ASM*, *AJ881UB.ASM* and *AJ881UC.ASM*. Assemble these modules and include the resulting object modules in your link specification file.

Miscellaneous Name Changes

The AMX 386/EP real mode code segment *AMXCODR* is of class 'CODE', not 'CODR'. The AMX 386/EP real mode data segment *AMXDATR* is also of class 'CODE', as it always was.

The end of real mode code and data is indicated by AMX 386/EP symbol *cj_psedata* (or *_cj_psedata*), not *AMREND*.

Calls to the following AMX 386 v1 PC Supervisor real mode procedures must be converted to call the AMX 386/EP equivalents. These conversions must be done by editing your real mode application source modules, if any, which reference the procedures. The names *cj_psxxxx* may be of the form *_cj_psxxxx* to conform to your C compiler's naming conventions.

AMX 386 v1	AMX 386/EP
<i>AMRINT</i>	<i>cj_psrint</i>
<i>AMRINX</i>	<i>cj_psrinx</i>
<i>AMRIOC</i>	<i>cj_psrioc</i>
<i>AMRIOX</i>	<i>cj_psriox</i>
<i>AMPCALL</i>	<i>cj_pspcall</i>
<i>AMSCALL</i>	<i>cj_psscalle</i>

The Restart Procedure *AMRDBRK* used to accommodate the manner in which some debuggers operate at breakpoints has been renamed *cj_psrdbrk*.

PC Supervisor Fatal Error Codes

The AMX 386/EP PC Supervisor fatal error codes are defined in header file *CJ814PE.DEF*. All of these fatal errors are funnelled through the AMX Fatal Exit Procedure *cjksfatal* in module *CJ814UF.C*. The fatal error condition is then reported on the PC video display as described in Chapter 4.8 and Appendix B of the PC Supervisor Reference Manual.

It is important to note that the PC Supervisor fatal error codes defined in header file *CJ814PE.DEF* are translated to AMX fatal error codes before entry to procedure *cjksfatal*. The revised error codes form a linear extension of the AMX fatal error codes listed in Appendix B of the AMX User's Guide.

PC Supervisor Configuration Module

With AMX 386 v1, the assembly language AMX System Configuration Module included all of the components necessary to meet the needs of the PC Supervisor, including real mode code and data segments. Since the AMX 386/EP System Configuration Module is coded in C, the real mode segment requirements cannot be met. Consequently, AMX 386/EP requires a separate assembly language PC Supervisor Configuration Module.

The PC Supervisor Configuration Module is created from your User Parameter File. Once you have converted your User Parameter File *CFGVC.UP* as described in Chapter 2.2 of this manual, you can generate your PC Supervisor Configuration Module using the AMX Configuration Manager. Select the PC Supervisor Configuration Module from the list of modules and click on the Generate button to create file *CFGVC.PCS*. Rename this file *CFGVC.ASM*, assemble it and link it with your application as described in the AMX 386/EP Tool Guide for the tools which you are using.

Configuration Manager Enhancements

The AMX Configuration Manager has been enhanced to allow you to define all of the configuration parameters formerly entered by appending the *...EXT* keyword statement to your User Parameter File.

Using the AMX Configuration Manager, edit your User Parameter File and select the PC Supervisor Configuration Module from the list of modules. The number of real mode Interrupt Stacks can be entered on the PC Supervisor property page. The Breakpoint Manager is selected and its entry and exit delays are defined on the Breakpoint property page. The *...EXT* linker and debugger attributes fields are no longer required.

Protected Mode Interrupt Stacks

AMX 386/EP requires one or more Interrupt Stacks for its use in servicing interrupts which occur while in protected mode. This process is described in detail in Chapter 2.3 of the PC Supervisor Reference Manual.

When you create or edit your User Parameter File using the AMX Configuration Manager, the number of protected mode Interrupt Stacks will default to four. You can adjust this number to meet your requirements by editing its value on the System property page.

Asynchronous Communications Adapter Device Driver

The Asynchronous Communications Adapter device driver has been enhanced to support up to 16 serial I/O lines (*COM1* to *COM16*). The new features listed below are described in Chapters 3.5 and 5.5 of the PC Supervisor Reference Manual.

- Transmit and receive timeouts
- Control over modem signal prerequisites for transmit and receive
- Improved control over line status:
 - sense and clear XON/XOF condition
 - resume transmission if blocked by XOF with no XON
 - read transmit and receive buffer character counts
 - flush transmit and receive buffers
 - disable transmission and reception

By default, the Asynchronous Communications Adapter device driver no longer requires *DSR* to receive or *DSR* and *CTS* to transmit. Hence, the driver will function with a simple 3-wire connection. If you want the new driver to mimic the AMX 386 v1 PC Supervisor driver, you must condition the serial line to require *DSR* and *CTS* as prerequisites. To do so, a task must make the new extended BIOS INT 20 (14H) function *84H* call prior to using the serial line.

PCS Device Configuration Module

The AMX 386/EP PC Supervisor Device Configuration Module *CJ814PAC.ASM* is constructed from the device parameter file *CJ814PAC.UP* and the Device Configuration Template file *CJ814PAC.CT*. These files replace their *AA882PCF.** counterparts.

If you altered your AMX 386 v1 Device Configuration Module to meet your specific serial and parallel device requirements, you will have to edit the new device parameter file *CJ814PAC.UP* and rebuild your Device Configuration Module. This procedure is described in Appendix E.1 of the PC Supervisor Reference Manual.

You will now be able to add support for up to sixteen serial lines if you so desire without having to edit any PC Supervisor source code.

You will observe that the *...PPA* descriptions for the parallel ports include an additional parameter *LOOPC* which you must define.

Appendix A. AMX C Code Conversion

AMX xxx to AMX vc C Code Conversion

xxx Name	vc Name	Notes	xxx Name	vc Name	Notes
General Control			Message Exchange Tasks		
ajentr	cjkslaunch			cjtkmxid	Get message exchange id
ajexit	cjksleave			cjtkmxinit	Initialize task
<input type="checkbox"/> ajfatl	cjksfatal			cjtkxdelete	Delete task
ajhook	cjkshook			cjtkxkill	Kill task
<input type="checkbox"/> ajprvl	cjkspriv		Interrupt Control		
<input type="checkbox"/> ajprvr	cjkspriv		ajint		No equivalent
ajupt		Handled by cjkslaunch	ajinx		No equivalent
<input type="checkbox"/> ajver	cjksver	Version info differs	ajispm		No equivalent
	cjkserror	User Error Procedure	ajidt---	cjksi---	Processor dependent
			ajitrp	cjksitrap	Processor dependent
			ajivt---	cjksi---	Processor dependent
Task Services (General)			Timing Control		
<input type="checkbox"/> ajend	cjtkend		ajclk		No equivalent
<input type="checkbox"/> ajresum	cjtkresume		<input type="checkbox"/> ajtick	cjtmtick	
<input type="checkbox"/> ajsusp	cjtksuspend		<input type="checkbox"/> ajtmcnv	cjtmconvert	
■ ajtkcre	cjtkcreate		<input type="checkbox"/> ajtmcre	cjtmcreate	
■ ajtkdel	cjtkdelete		<input type="checkbox"/> ajtmdel	cjtmdelete	
<input type="checkbox"/> ajtkid	cjtkid		■ ajtmrd	cjtmread	
■ ajtkill	cjtkkill		■ ajtmtag	cjksfind	
<input type="checkbox"/> ajtkpry	cjtkpriority		<input type="checkbox"/> ajtmwr	cjtmwrite	
<input type="checkbox"/> ajtkstp	cjtkstop		<input type="checkbox"/> ajtslv	cjtmslice	
■ ajtksts	cjtkstatus		<input type="checkbox"/> ajtsof	cjtmsopt	
■ ajtktag	cjksfind		<input type="checkbox"/> ajtson	cjtmsopt	
<input type="checkbox"/> ajtk tcb	cjkt tcb		Time/Date Manager		
<input type="checkbox"/> ajtktrm	cjtkterm		<input type="checkbox"/> ajtdf	cjtdfmt	
<input type="checkbox"/> ajtrig	cjtktrigger		<input type="checkbox"/> ajtdg	cjtdget	
			<input type="checkbox"/> ajtds	cjtdset	
Task Message Passing			Semaphore Manager (counting)		
■ ajgmsg		Use message exchange	<input type="checkbox"/> ajsmcre	cjsmcreate	
■ ajsend		Use message exchange	<input type="checkbox"/> ajsmdel	cjsmdelete	
■ ajsendp		Use message exchange	<input type="checkbox"/> ajsmget	cjsmwait	
■ ajsenw		Use message exchange	<input type="checkbox"/> ajmsmsig	cjsmsignal	
■ ajsenwp		Use message exchange	■ ajmsmtag	cjksfind	
■ ajwakc	cjtkmsgack	For mailboxes or	<input type="checkbox"/> ajsmwat	cjsmwait	
■ ajwakcs	cjtkmsgack	message exchanges		cjsmstatus	Semaphore status
Task Timing & Synchronization			Semaphore Manager (resource)		
<input type="checkbox"/> ajwait	cjtkwait		<input type="checkbox"/> ajsmcre	cjsmcreate	
<input type="checkbox"/> ajwake	cjtkwake		<input type="checkbox"/> ajsmdel	cjsmdelete	
<input type="checkbox"/> ajwapr	cjtkwaitclr		<input type="checkbox"/> ajsmfre	cjrmfree	
■ ajwatm	cjtkwaitm		<input type="checkbox"/> ajsmrls	cjrmrls	
	cjtkdelay	Delay for interval	<input type="checkbox"/> ajsmrsv	cjrmrsv	
Task Signals				cjrmstatus	Resource status
■ ajsgnl		No equivalent			
■ ajsgrd		No equivalent			
■ ajsgres		No equivalent			
■ ajsgwat		No equivalent			

AMX xxx to AMX vc C Code Conversion

xxx Name	vc Name	Notes	xxx Name	vc Name	Notes
Event Manager			Memory Manager		
<input type="checkbox"/> ajevcre	cjevcreate		<input type="checkbox"/> ajmau	cjmmuse	
<input type="checkbox"/> ajevdel	cjevdelete		<input type="checkbox"/> ajmfre	cjmmfree	
<input type="checkbox"/> ajevnt	cjevwaits		<input checked="" type="checkbox"/> ajmgeh	cjmmget	
<input type="checkbox"/> ajevrd	cjevread		<input checked="" type="checkbox"/> ajmget	cjmmget	Requires memory pool id
<input type="checkbox"/> ajevsg	cjevsignal	Event pulse allowed	<input type="checkbox"/> ajmgsz	cjmmsize	
<input checked="" type="checkbox"/> ajevtag	cjksfind		<input checked="" type="checkbox"/> ajmhan	cjmmcreate	Returns pool id
<input type="checkbox"/> ajevwat	cjevwait		<input checked="" type="checkbox"/> ajmset		No equivalent
	cjevstatus	Event group status		cjmmdelete	Delete a memory pool
				cjmmid	Get a memory block's memory pool id
Message Exchange Manager				cjmmsection	Add a memory section to a memory pool
<input type="checkbox"/> ajmxcre	cjmxcreate		Linked List Manager		
<input type="checkbox"/> ajmxdel	cjmxdelete		<input type="checkbox"/> ajlcre	cjlmcreate	
<input type="checkbox"/> ajmxget	cjmxwait		<input type="checkbox"/> ajlhead	cjlmhead	
<input type="checkbox"/> ajmxsnd	cjmxsend		<input type="checkbox"/> ajlinsc	cjlminsc	
<input checked="" type="checkbox"/> ajmxsndp		Pass by reference only	<input type="checkbox"/> ajlinsh	cjlmminsh	
<input checked="" type="checkbox"/> ajmxtag	cjksfind		<input type="checkbox"/> ajlinsk	cjlminsk	
<input type="checkbox"/> ajmxwat	cjmxwait		<input type="checkbox"/> ajlinst	cjlmint	
	cjmxflush	Flush message exchange	<input type="checkbox"/> ajlmerg	cjlmmerg	
	cjmxstatus	Message exchange status	<input type="checkbox"/> ajlnext	cjlmnext	
Mailbox Manager			<input type="checkbox"/> ajlordk	cjlmordk	
	cjmbcreate	Create mailbox	<input type="checkbox"/> ajlprev	cjlmprev	
	cjmbdelete	Delete mailbox	<input type="checkbox"/> ajlrmvc	cjlmrmvc	
	cjmbflush	Flush mailbox	<input type="checkbox"/> ajlrmvh	cjlmrmvh	
	cjmbsend	Send to mailbox	<input type="checkbox"/> ajlrmvt	cjlmrmvt	
	cjmbstatus	Mailbox status	<input type="checkbox"/> ajltail	cjlmntail	
	cjmbwait	Wait on mailbox	Circular List Manager		
Buffer Manager			<input type="checkbox"/> ajabl	cjclabl	
<input checked="" type="checkbox"/> ajbau	cjbmuse		<input type="checkbox"/> ajatl	cjclatl	
<input checked="" type="checkbox"/> ajbcre	cjbmcreate		<input type="checkbox"/> ajrbl	cjclrbl	
<input type="checkbox"/> ajbdel	cjbmdelete		<input type="checkbox"/> ajrstl	cjclinit	
<input checked="" type="checkbox"/> ajbfre	cjbmfree		<input type="checkbox"/> ajrtl	cjclrtl	
<input type="checkbox"/> ajbget	cjbmget	Wait with timeout allowed	C Services		
<input checked="" type="checkbox"/> ajbsz	cjbmsize		<input type="checkbox"/> ajdi	cjcfdi	
<input checked="" type="checkbox"/> ajbia		Delete and recreate pool	<input type="checkbox"/> ajei	cjcfci	
<input checked="" type="checkbox"/> ajbip		Delete and recreate pool	<input type="checkbox"/> ajin--	cjcfin--	
<input checked="" type="checkbox"/> ajbtag	cjksfind		<input type="checkbox"/> ajout--	cjcfout--	
	cjbmid	Get buffer's pool id	ajproc, ajprocq		
	cjbmstatus	Buffer pool status	ajsint, ajsintq		
			ajmodl		
			ajgofs, ajgseg, ajgsreg		
			ajsofs, ajsseg, ajssreg		

Appendix B. AMX Design Constants

B.1 AMX 386/ES Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX 386/ES design constants are summarized below.

Characteristic	AMX 386/ES	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	256	128	128	256
Minimum Interrupt Stack (bytes)	256	128	256	256
Minimum task storage (bytes)	512	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	no	yes	yes	n/a
DOS Command Task	no	yes	no	n/a
File Manager	yes	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	little	little	little	big
Memory model	Note 2	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 and 386/ES support the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

n/a: Not applicable

B.2 AMX 386/EP Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX 386/EP design constants are summarized below.

Characteristic	AMX 386/EP	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	256	128	128	256
Minimum Interrupt Stack (bytes)	256	128	256	256
Minimum task storage (bytes)	512	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	yes	yes	yes	n/a
DOS Command Task	no	yes	no	n/a
File Manager	Note 4	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	little	little	little	big
Memory model	Note 2	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 and 386/EP support the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

Note 4: AMX 386/EP provides access to DOS file services.
The Phar Lap TNT DOS-Extender is a prerequisite for AMX 386/EP.

n/a: Not applicable

B.3 AMX 68000 Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX 68000 design constants are summarized below.

Characteristic	AMX 68000 v3	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	256	128	128	256
Minimum Interrupt Stack (bytes)	256	128	256	256
Minimum task storage (bytes)	512	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	n/a	yes	yes	n/a
DOS Command Task	n/a	yes	no	n/a
File Manager	yes	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	big	little	little	big
Memory model	flat	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 supports the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

n/a: Not applicable

B.4 AMX CFire Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX CFire design constants are summarized below.

Characteristic	AMX CFire	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	256	128	128	256
Minimum Interrupt Stack (bytes)	256	128	256	256
Minimum task storage (bytes)	512	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	n/a	yes	yes	n/a
DOS Command Task	n/a	yes	no	n/a
File Manager	yes	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	big	little	little	big
Memory model	flat	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 supports the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

n/a: Not applicable

B.5 AMX PPC32 Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX PPC32 design constants are summarized below.

Characteristic	AMX PPC32	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	1024	128	128	256
Minimum Interrupt Stack (bytes)	1024	128	256	256
Minimum task storage (bytes)	1024	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	n/a	yes	yes	n/a
DOS Command Task	n/a	yes	no	n/a
File Manager	yes	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	big (Note 4)	little	little	big
Memory model	flat	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 supports the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

Note 4: Supports both big and little endian models. Most common is big endian.

n/a: Not applicable

B.6 AMX 4-ARM, 4-Thumb Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX 4-ARM and AMX 4-Thumb design constants are summarized below.

Characteristic	AMX 4-ARM AMX 4-Thumb	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	256	128	128	256
Minimum Interrupt Stack (bytes)	256	128	256	256
Minimum task storage (bytes)	512	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	n/a	yes	yes	n/a
DOS Command Task	n/a	yes	no	n/a
File Manager	yes	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	little (Note 4)	little	little	big
Memory model	flat	Note 1	Note 2	flat
Runs in privileged mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 supports the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

Note 4: Supports both big and little endian models. Most common is little endian.

n/a: Not applicable

B.7 AMX MA32 Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX MA32 design constants are summarized below.

Characteristic	AMX MA32	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	512	128	128	256
Minimum Interrupt Stack (bytes)	512	128	256	256
Minimum task storage (bytes)	1024	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	n/a	yes	yes	n/a
DOS Command Task	n/a	yes	no	n/a
File Manager	yes	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	little (Note 4)	little	little	big
Memory model	flat	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 supports the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

Note 4: Supports both big and little endian models. Most common is little endian.

n/a: Not applicable

B.8 AMX 386/ET Design Constants

AMX uses a set of design constants which vary according to the constraints imposed by the target processor. The AMX 386/ET design constants are summarized below.

Characteristic	AMX 386/ET	AMX 86 v3	AMX 386 v1	AMX 68000 v2
Size of integer (bytes)	4	2	4	4
AMX error codes	<i>int</i>	<i>int</i>	<i>int</i>	<i>int</i>
Event group flags per group	32	16	32	32
AMX id size (bytes)	4	2	4	4
Maximum number of tasks	Note 3	100	100	100
Minimum AMX message (bytes)	12	12	12	12
Default AMX message (bytes)	12	12	12	12
Minimum number of envelopes	10	10	10	10
Maximum number of envelopes	Note 3	<4096	Note 3	Note 3
Minimum Kernel Stack (bytes)	256	128	128	256
Minimum Interrupt Stack (bytes)	256	128	256	256
Minimum task storage (bytes)	512	128	128	256
Minimum buffer size (bytes)	8	8	8	8
Maximum buffer pool (bytes)	Note 3	64k	Note 3	Note 3
Minimum memory block (bytes)	16	16	16	16
Minimum memory section (bytes)	128	64	76	96
PC Supervisor	no	yes	yes	n/a
DOS Command Task	no	yes	no	n/a
File Manager	yes	DOS	DOS	no
Terminal Handler	no	no	no	yes
Endian model	little	little	little	big
Memory model	Note 2	Note 1	Note 2	flat
Runs in supervisor (kernel) mode	yes	n/a	yes	yes
AMX coded in C or assembler	C	asm	asm	asm

Note 1: AMX 86 supports the Large model and the Medium model with *FAR* pointers.

Note 2: AMX 386 and 386/ET support the 32-bit *NEAR* (flat) model with *FAR* pointers.

Note 3: Limited only by available memory

n/a: Not applicable