1. Introduction

1.1. About

Welcome to Chaos Assembler 3... We hope you'll have a great time using it and we did our best to make the program usage as easy as possible... Also we tried to keep this documentation as small as possible... Nevertheless you'll have to sit down for a while, read this docs, set your preferences and get used to the interface of Chaos Assembler 3. This process will not take too long, but you'll be astonished about what this program has to offer comparing it to other assembler IDE's (some well known are of course WBASS and COMPASS). Chaos Assembler of course has one big disadvantage, it's not an MSX program. Nevertheless, it can be used to produce great products.
Hopefully it's not too much for you...

We hope you really enjoy using Chaos Assembler 3 as much we enjoyed making it...

Suggestions, bugs, ideas and other things: info@TeddyWareZ.cjb.net

1.2. A little bit history

In 1997 (already over 4 years ago!) some guy named d-fader thought it was necessary to learn Z80 ASM code. So he tried and tried and he even got better and better. After a few months he thought he was quite good. Things he couldn't do at that time was e.g. making a scroll on MSX.
He once in the summer of 1997 visited a friend to do some computer related stuff. When he was there, his brother interrupted the conversation and asked if he already coded MSX Z80 ASM. He knew him from a few years before that, but their contact somehow broke. As he asked, d-fader answered "Sure! Though I still can't make a good scroll in ASM", so the brother (from now on identified as HeXx) said: "Aaaah! It's been a while for me, but it's quite easy: Just do this and then this, after that do that and then it should work". He talked about bit shifting, anding stuff, also he talked about some MSX2 copy routine and such stuff...
So d-fader answered: "Hmmm... Ehmz.... eeh... Okay!". HeXx KNEW d-fader didn't understand a word he said. So he said to d-fader: "Tomorrow, 8:30, you WILL be here. See you tomorrow." and he left. As you probably can imagine, d-fader was quite astonished about what he said... He had gone home, not understanding and the next day, as HeXx said, he was there at 8.30. HeXx opened the door. He started: "We're gonna make a scroll. I'm NOT gonna learn you MSX ASM, I'm just gonna give you a little push in the right direction".
So the two got the dusty VG8245 of HeXx, plugged it in, started it, started WBASS2 and they were ready to go. The last time HeXx even started WBASS2 was about 6 years (!) before that! After that he said okay. Now... First a great thing of MSX2, he started.

HeXx : You know Metal Gear right?
d-fader : Yes.
HeXx : You also know Metal Gear is screen 5?
d-fader : Yes.
HeXx : Ever asked yourself how come Metal Gear builds up its screens so damn fast?
d-fader : No. Fast? Hmm. You're right! A normal 'copy' command is quite slow!
HeXx : Exactemondo.
HeXx : And that has a reason. The VDP of the MSX can copy REALLY fast! Just not by using the normal 'copy' routine!
d-fader : Wow! How?
HeXx : That's what we're gonna find out now!

At that time we didn't have any docs, no VDP docs, no nothing, just some MSX1 Z80 machine code book. He said we were gonna look for a magazine where the 'fast' copy routine was explained. After a while looking they found it and they started coding. Actually d-fader just looked how HeXx coded :D.
As it was 6 years HeXx didn't do Z80, he forgot a whole lot, didn't have any significant sources anymore and the sources he still had, had nothing to do with MSX2 copy routines. So they (together) made the copy routine. After about a day they finished it and HeXx was back in business, he could code Z80 again. It had all came back. The days after (wednesday-friday) they coded on the scroll routines. It was a really HOT week. 35 degrees every day and no ventilation. But they coded from 8.30AM to 5PM every day.
Friday they had a working scroll with some star background and vdp scroll in it. That day, the guy now known as Chaos came back from vacation and d-fader would go the week after that (starting on Saturday). So d-fader visited Chaos that night and he showed the demo. Chaos just was amazed! He asked HOW'D you do THAT??
D-fader told the HeXx story (and the calls HeXx would made at 8.15AM to check if d-fader already was out of bed :D). Chaos didn't know what to say.

D-faders week of vacation followed and after that he returned to HeXx, which had coded even more stuff... He was really liking it! :D

As they discussed d-faders vacation and HeXx his code stuff, d-fader came up with the idea to form a new MSX demo group. The intention of this group would be that they would've been quite different than other MSX groups in a good way. As HeXx liked the idea a lot, they soon came up with the question what the name of the group should be... At that moment the brother of HeXx came in the room where they were and he started thinking of a name. After a while he already came up with several ideas. One of them was TeddyWareZ. Though at first they thought it was not the best idea, after a short while they came to the conclusion that TeddyWareZ was the best idea and they already started to like it...

So far so good. A new group was born, though no one else knew it... After that the group started to develop. They would make a first demo-disk which would've been 50% ASM. After 1 and a half year of developing it was finished. Teddy's In Action was released at Zandvoort '98. Though we had a stand, it was more like a wooden table with some computers on it. Visitors of the fair didn't

even see us, the only thing they saw were the big stands of FD and such. At about 4PM (one hour before the fair would end) everybody had seen all the big stands and everybody walked up to us looking at us questioning who we were... We demonstrated TIA whole day, nobody seemed to care but after 4PM everybody came to our stand buying TIA.
We also gave most of the dutch magazines a review disk and TeddyWareZ was known to the public!

Now you might ask, what has this to do with the product I'm now reading the help of? The answer is quite simple, I'm gonna get to that right now. Teddy's In Action was created using WBASS2. A great assembler / IDE, the difference with most (all?) other groups at that time was that we didn't use WBASS on MSX. We used an MSX emulator (MSX4PC) and in that emulator we used WBASS to create the complete disk! Also we've used Graph Saurus with this emulator. We really liked this approach. To make a long story short, we almost quit MSX after the release of TIA. At that time d-fader wanted to continue on the TI-83 (calculator of Texas Instruments) scene, which also has a Z80 processor. As d-fader started looking for an assembler he thought of maybe using the editor he would find for MSX.
He found a REALLY neat wrapper for TASM (Telemarks cross ASseMbler) called Chaos Assembler 2 (CA2). The rest of the products TeddyWareZ has released until now, are written using CA2 (this includes USW and SCC-Blaffer NT)!

Though CA2 was a great wrapper, it was actually a wrapper for the TI-83 calculators and so it had nothing to do with MSX. Soon raise the question if CA2 could be adapted for MSX usage. At a certain time we contacted the author of CA2, and he even replied and he was willing to adapt it for MSX usage! As time passed by we lost contact with him and d-fader started developing Win32 applications and then the idea came to him to make an CA2 for MSX usage. The intention was to create a program with the same options as CA2, but as time passed by, the product became more and more extensive. That's when the idea came to call it Chaos Assembler 3.

Chaos Assembler 3 has most options of CA2 (and thus also we used TASM to compile your sources), but CA3 has about a zillion options more!

TeddyWareZ is VERY proud of this product and we hope this product will fit the MSXers in coding there own projects together.

Chaos Assembler 3... Try it, use it, LOVE it!

have fun!

The TeddyWareZ crew.

1.3. Why Chaos Assembler 3?

Why Chaos Assembler 3?

Well.. Guess that is a pretty easy question. This project started out of frustration that CA2 (Chaos Assembler 2) did not support any needs of the MSX system. These include a cool image viewer, project support and of course a sprite editor. So I started coding and thought up a number of other cool options for the program and built them in. Therefor the program has become much more advanced than CA2.

Also this project was made for experience purposes and because we love MSX.

1.4. Disclaimer

This is a pretty standard issue... :)

TeddyWareZ or anyone affiliated with us cannot be held responsible for any hardware, software, mental, physical or any other form of damage resulting from use of this product. We do not encourage any illegal, immoral or irresponsible use or misuse of anything or anyone associated with us and/or this product. All procedures described in this TeddyWareZ product are executed by the user his/her own risk.

2. Overview

2.1. IDE, Wrapper, Compiler, HELP!

In the introduction I was throwing with some difficult words. These words aren't that difficult, but I think it clears up a lot of confusion just to explain how CA3 actually works...
An IDE is a shortening of Integrated Development Environment. This means it's an application which has an editor and a compiler in one. Good examples of IDE are WBASS and COMPASS...
The word wrapper is actually the same, the only difference is that a wrapper uses an external compiler to compile sources made in the editor. It's kind of an 'wrap around' the compiler program. A good example of a wrapper is Chaos Assembler 2.
The compiler is the base of programming. A compiler is used BOTH in an IDE and in a wrapper. The compiler makes binary (compiled) data of sources typed in some kind of editor. A good example of this approach is GEN80. GEN80 users mostly use TED (a dutch Text EDitor) to create ASCII sources (which of course can be created by any text editor which can save it files as an ASCII file).

What is the best approach and why?

All three approaches have both advantages as also disadvantages.

1. IDE
IDE has one big advantage. Because the compiler is IN the application you won't have to use an external compiler and the source you try compile will always be compiled, as it's the same program that edits and compiles. The big disadvantage of the IDE approach is that you have to have written the compiler yourself or you should have the source of the compiler and also have to know how it works.

2. Wrapper
Wrappers also have a big advantage. Because it is external, the compiler can change version and the wrapper (which didn't change of version) still can handle the new compiler. This of course also comes in handy because at that moment only the compiler has to be replaced, this is quite handy when having HTTP or e-mail transfer in mind. A wrapper also has one big disadvantages. The wrapper has to call the compiler correctly, this means the wrapper application has some settings for using the compiler. Also when a compiler changed its interpretation in a certain way, sources could have errors all of a sudden when compiling on another computer (which could have another version of the compiler). Also a wrapper needs to GRAB the output of the compiler and cannot fully interact with the written source. A wrapper is 'further' away from the source than an IDE.

3. Compiler
Compilers have an advantage. Any kind of editor can be used to make binary files. The great disadvantage of this is that no editor will actually interact with the user and this means no auto calling of the compiler, no result output in the editor and so on...

CA3 uses the second approach (the wrapper). CA3 uses TASM (Telemarks cross ASseMbler). TASM is a really extensive compiler which can handle all kinds of neat coding styles. This means e.g. that CA3 can detect errors (generated by TASM) and list them after you compiled a file.
All those options and features will be explained later on in this help file.

2.2. Main items

2.2.1. The main window

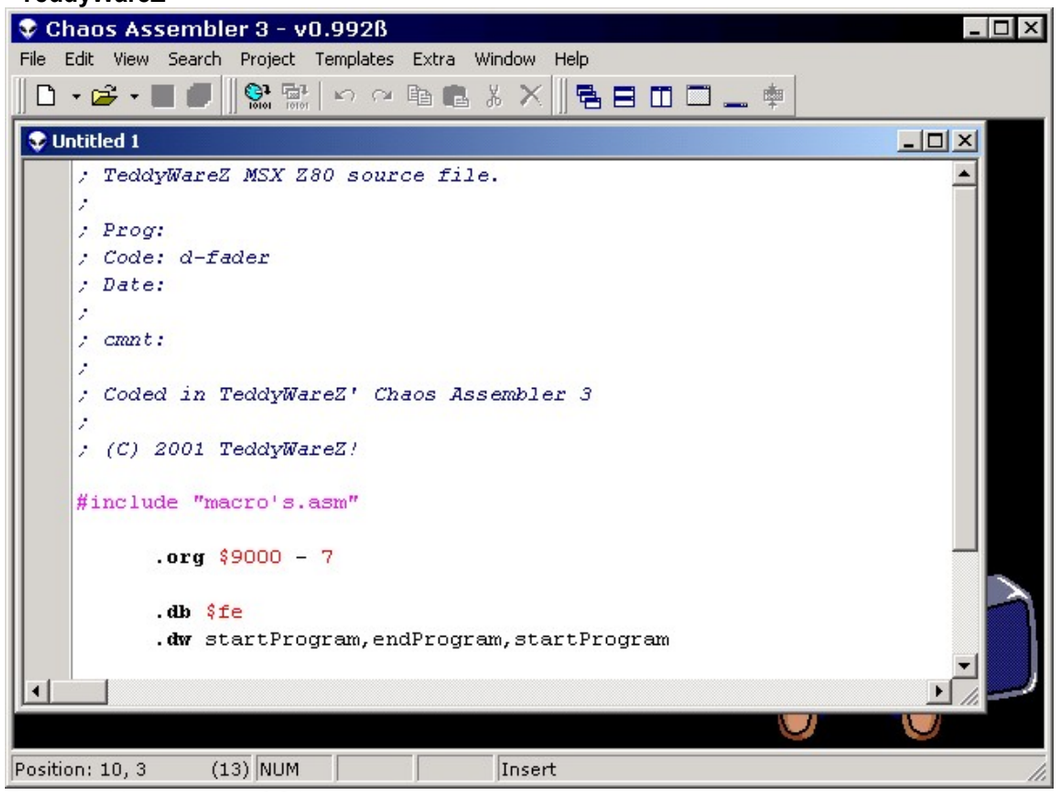

Figure 1. Main screen of CA3

Here you see the main screen of Chaos Assembler 3. What always will be visible is the **main menu** bar at the top of the screen as also the **toolbars** beneath that. These are the key features of CA3. With these two items you can go everywhere in the program. As you might notice, there's a window inside the main window. That's what we call a **child**. Automatically the main window will be called the **parent**.

The child window you see here is the child window you will use the most. In the child you see, you are going to edit your source. This child (further referred to as 'the edit child') has really neat options build in to assist you as a programmer. More about this edit child in the next chapter.

The main menu and the toolbars are very related to each other. The most often used items in the main menu you find back at the toolbars. The first one e.g. is a little blank white paper. When you click on that button, a new source file will be generated and immediately an edit child will appear.

You can open as many child windows as you like. There are more windows of this kind in CA3.

The **image viewer** and the **sprite editor** e.g. have the same characteristics as the edit child.

2.2.2. The edit child

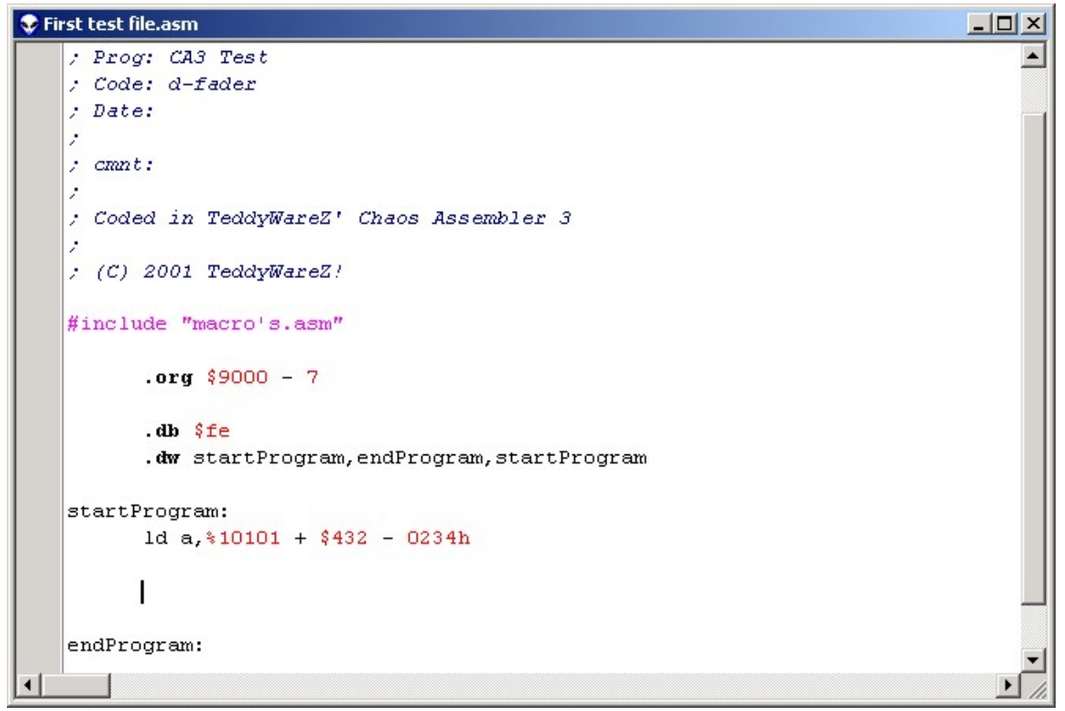

Figure 2. The edit child

A small introduction to the edit child

Before I start explaining what the edit child all can do, I first want to tell that you should realize that this part of the product is REALLY important for you. Not just because we say so, I say this because I **KNOW** so. There has been put hours and hours of work to complete this editor and it REALLY fits the MSX programmers needs.

- TeddyWareZ -

What you see here is not just some editor. Maybe you notice the coloring and changing font styles of some words in the editor. That's one of the extensions to a regular editor. It is tuned to use with Z80 code and specifically the way Z80 code is interpreted by the compiler used with CA3 (TASM).

This is just a small extra feature of a default editor. What make this editor so special are the so-called **code completion** and **code tool tips**. To keep it short, code completion KNOWS what you are doing. E.g. when you type **call startP** it can drop down a menu which will contain all labels used in the source file (or an included file) and will try to find out if there's a label starting with **startP**. In this case that's true (label: startProgram) and the drop down menu will automatically select the label **startProgram**. When you then press enter, code completion will automatically insert **startProgram** after 'call'. That's the basis of **code completion**.

Code tool tips is something totally different, but equal as cool... Ever been in a situation where you couldn't recall if the command you typed was correct, or the parameter you used was correct Z80 language? Code tool tips can help you with that. When you type **ld a,** in a source and give a call to the code tool tips, this will show ALL possibilities of **ld a,**!

Those are the most significant additions to a 'normal' editor. More about these features will follow later on in this document.

3. Using Chaos Assembler 3

3.1. First contact

When you start Chaos Assembler 3 for the first time you will be warped directly to the settings screen. In this screen you can (and have to) set your preferences for Chaos Assembler 3. All settings will be set to default, those are the best while developing the application. Nevertheless, your taste of 'perfection' might be a whole lot different than ours. That's what's the settings screen is for.

Because of the fact that this is the first actual screen you will see when you start using Chaos Assembler 3, I'll explain the complete settings screen first.

3.2. The settings screen

3.2.1. In general

The settings screen of CA3 is used to set all your preferences in CA3. In general I would like to note what all buttons do you can see at the bottom of the settings screen...

Register file types
The button with the caption register file types will register the **.asm** and **.cap** file types in the windows explorer. This means that if these file types are registered and you would double-click a .asm or .cap file in the windows explorer, CA3 will be launched and the file you clicked on will be opened... You can see the files are registered in the windows explorer by the icon next to the file. If the icon looks like the icon you see at the top of this (and all other) topic, they're registered. If CA3 was already open, there will not be a new CA3 launched, but the file you clicked will be opened in the CA3 you have already opened. If the file types are not registered, you can press this button to register them...

OK
This button will save all changes you made and close the settings screen.

Cancel
This button will discard all changes you made and close the settings screen.

Apply
This button will apply all settings (i.e. Save all settings) but **won't** close the settings screen.

3.2.2. General tab

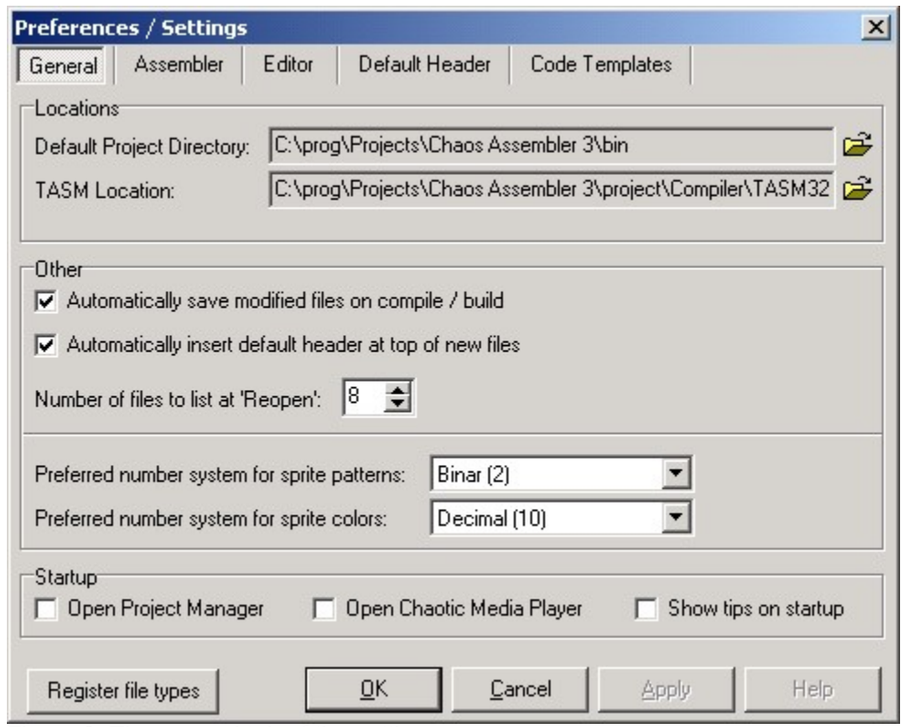

Figure 3. The general tab of the settings screen

The general tab is the first out of five tab sheets in which you can set all your preferences for CA3. There are pretty much options, so let's not waste any space on introducing the tabs, I'll head straight on to the explanation of all settings!

Locations

Default project directory

Click on the little folder on the right hand side of the location box to specify a 'default' project directory. The default project directory will be used for your convenience. When you e.g. for the first time save a file, CA3 will automatically jump to this directory so you won't have to search it every time. This can come in quite handy when you e.g. have a directory where you save all your source files under.This could be e.g. **C:\Programming\CA3\Projects**.

TASM Location

As CA3 is a so called wrapper (see Introduction chapter), CA3 needs to know which compiler to use. CA3 can use two different compilers. The 16-bit version of TASM and the 32-bit version of TASM. The difference between these two is that Windows NT (and thus also Windows 2000) cannot execute the 16-bit version of TASM pretty well. Mostly it crashes. That's why the 32-bit version is supported too. On the other hand, because the 32-bit version has to be called quite differently by CA3 than the 16-bit version, sometimes Windows 9x can't handle the 32-bit version that well... If you have any problems compiling your source, please refer to the **trouble shooting** section of this help file.

Other

Automatically save modified files on compile / build

This option is ***HIGHLY*** recommended and should only be turned off when you are sure you want it. This option (when enabled) saves all modified files of the project / assembly file (projects will be explained later) before calling the external TASM compiler. Why is this so necessary? Well, if you don't do this, TASM will used the last saved file to compile, even if you modified the file already. But because the file wasn't saved, TASM will compile the file you already had saved before, because it's an external compiler.

Automatically insert default header at top of new files

CA3 supports **default headers**. This will be explained later. A default header is a piece of text (code) that's pasted in a file when a new file is created. With this version you can enable or disable it. When disabled, the default header won't be inserted at the top of a new file...
This option is recommended.

Number of files to list at 'Reopen'

CA3 remembers all files you've ever opened, detailed information about this will follow later on in this document, and CA3 can 'reopen' these files again. This is done so you won't have to search for all the most recent file you've opened the last time you started CA3. This option let's you decide how many recent files will be shown. If you fill in 8 (default) this will mean that the 8 most recent files will be listed.

Default number system for sprite patterns

The WYSIWYG sprite editor (explained later) allows you to convert the sprites you make to Z80 ASM data structures so you can easily use the sprite in your source. A sprite of course consists of two parts. The first part is the pattern data and that data can be copied to the clipboard in three formats. First the binary, which is recommended, so you can actually see a pattern in your sprite.

Default number system for sprite colors

The second one of the data structures is the color data. The recommended use is decimal or hexadecimal, whatever you prefer. The binary system is not recommended because of two reasons, first of all it makes your code NOT readable and secondly, it's not natural to have color data in binary mode, cause changing the data is inviting for mistakes!

Startup

Open project manager

When enabled, the project manager will automatically show the minute you start CA3. The project manager will be explained later on in this help file.

Open Chaotic Media Player

When enabled, the Chaotic Media Player (CMP) will automatically show the minute you start CA3. The CMP will be explained later on in this help file.

Show tips on startup

When enabled, CA3 will show little '**Did you know**' sentences when you start CA3. This is a nice feature and is recommended for those who just start with CA3. You can learn some neat things about CA3 in this way without having to browse through this help file.

3.2.3. Assembler tab

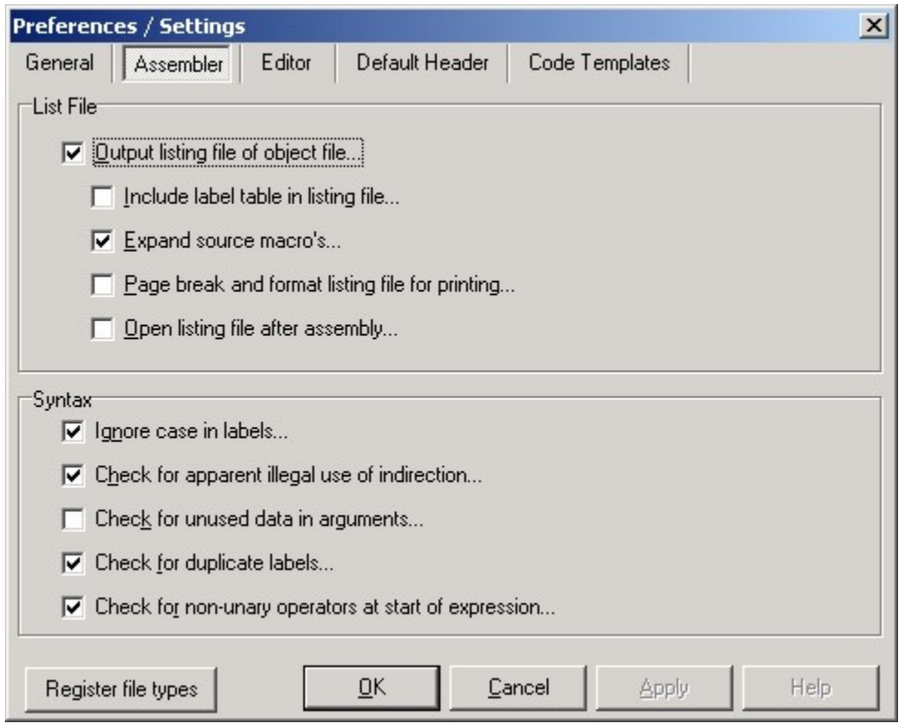

Figure 4. The assembler tab of the settings screen

Listings

Output listing file of object file
TASM generates an object file (the actual file that is runnable on the MSX) and besides that, you can direct CA3 to generate a listing file of the generated output object file. When enabled you have 4 extra options regarding to the listing file. These are explained in the **UNREGISTERED DEMO VERSION** part of the TASM documentation...

Syntax

Syntax options for the TASM compiler
Enable one or more of these options to let TASM check for these syntax related parameters. For more information about this is again refer to the **Invocation** part of the TASM documentation...

3.2.4. Editor tab

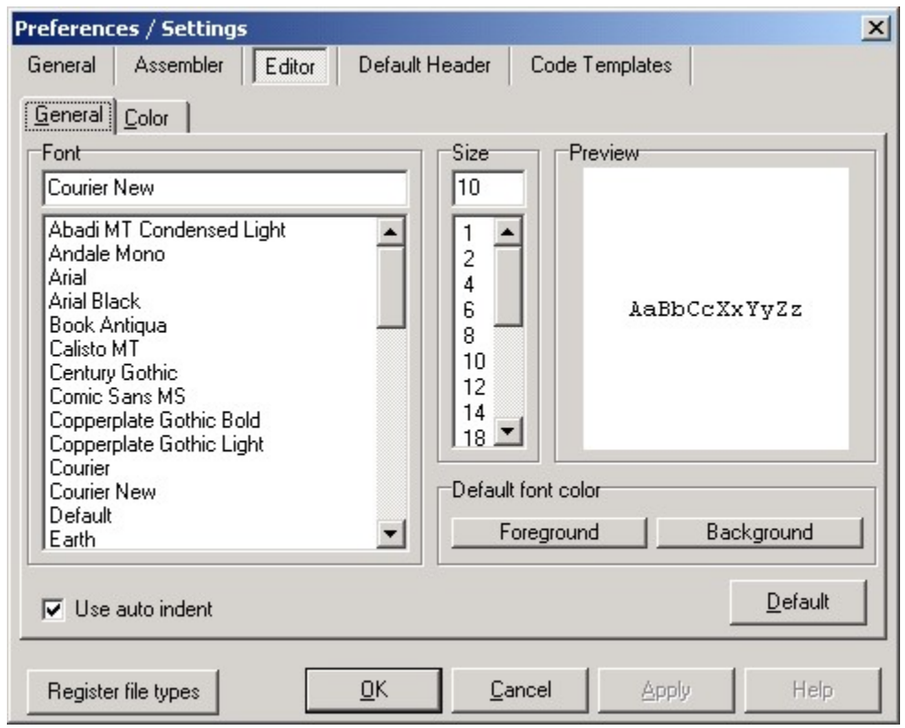

Figure 5. General tab of the editor section in the settings screen.

The editor section of the settings screen, let's you decide how the editor of CA3 (edit child) looks like. First the general tab sheet.

Font

Font
What kind of font do you want to use in the editor? Recommended is **COURIER NEW**. Be sure to at least use a FIXED font width. This is for the readability of your source, it just looks better with a fixed font and is better for your eyes since reading that will take not as much energy when every letter / word has a fixed space to each other.

Size
The size of the font you want to use. Recommended is 8-12. Try not to get the font size any larger, check the Font section why (readability).

Preview
When you change the font and or font styles, the changes will be directly applied to the preview box. In that way you can decide whether or not you like the current settings, before making the changes 'for real'.

Use auto indent
TASM requires every command to have at least one white space at the beginning of the line. Normally you use a **TAB** character for that. That's just for the readability. When you check this option, CA3 will enter the white spaces at the beginning of the line you pressed enter into the new line. With an example this is better to explain:

```
MyLabel:
  ld a,4
  ld b,6
  call MyProc [Enter]
  |
```

The | indicates the cursor position after you've pressed enter at the position where the **[enter]** text is. That's auto indent. Recommended!

Default font color
This is the color for the default font used in the editor. More about this in the Colors tab of this editor screen. Recommended font colors are **Black** and **White** (Black for foreground and White for background).

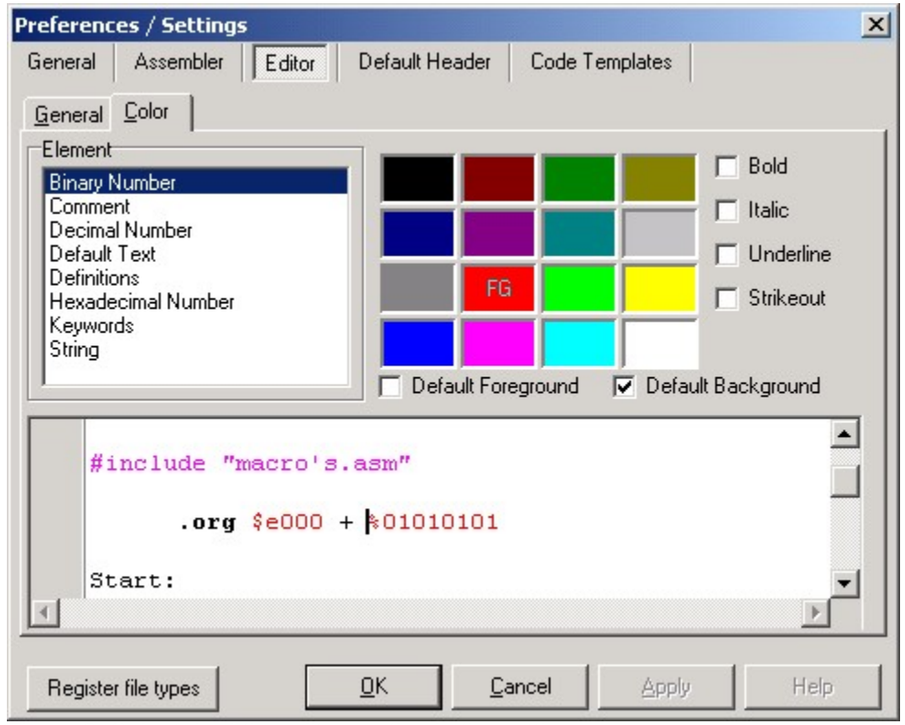

Figure 6. Color tab of the editor section in the settings screen.

Since the editor of CA3 has a syntax highlighting routine in it, we thought it would be nice to make these colors customizable. And that's what you can set right here.

Element

Select an element
Click on one of the listed elements to change the color of that element. As you can see here, I've selected the **binary number** element. Directly after you select an element, the little color palette will show the current colors and styles for the selected element. When you enable the **Default foreground** the color used for a binary number will be the color selected for the font in the General tab of this settings screen. The same goes for the **Default background** only then it (of course will be for the background color of the element).
Also you can select a font style to be used for the element. If you check the **.org** in the preview editor, you'll notice it's bold. That element has both the default fore- and background color, but has the **bold** option checked.

To select a certain color, use **LEFT MOUSE CLICK** for a foreground color and **RIGHT MOUSE CLICK** for a background color.

3.2.5. Default header tab

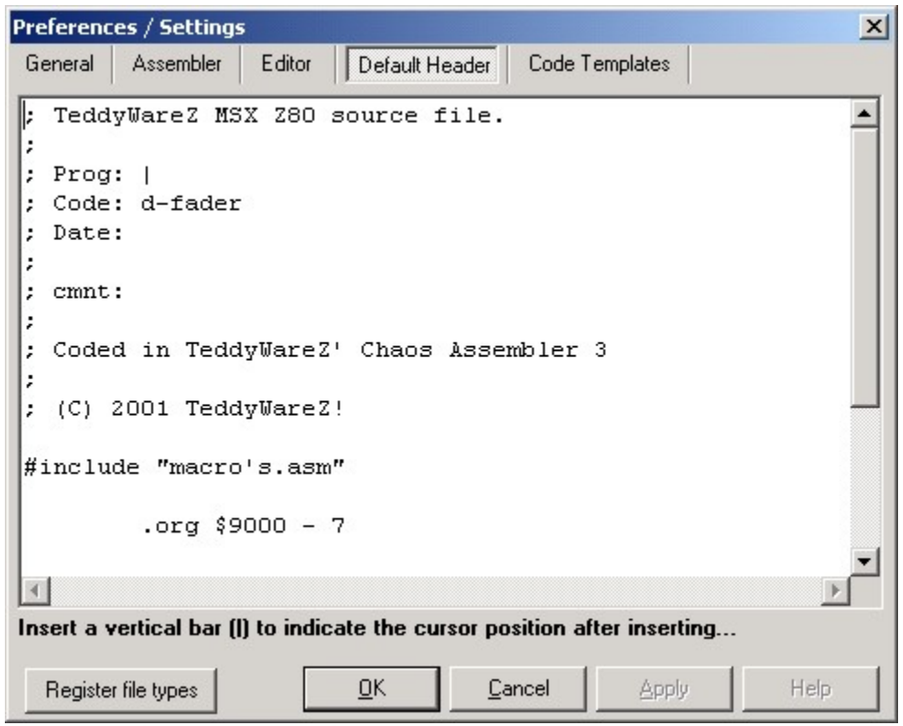

Figure 7. Default header tab of the settings screen.

What is a default header?

The default header is a nice feature used with new files. When you've enabled the option in the General tab (which is the default and also recommended) the piece of text you've entered here will be pasted at the top of a new file.

How do I use a default header?

This is quite easy, just type away in this window. There's no restriction in the size of the text, though you probably don't want it too big. When you use the pipe symbol (|) for the first time, the cursor will be placed at that point after inserting the default header.

3.2.6. Code Templates tab

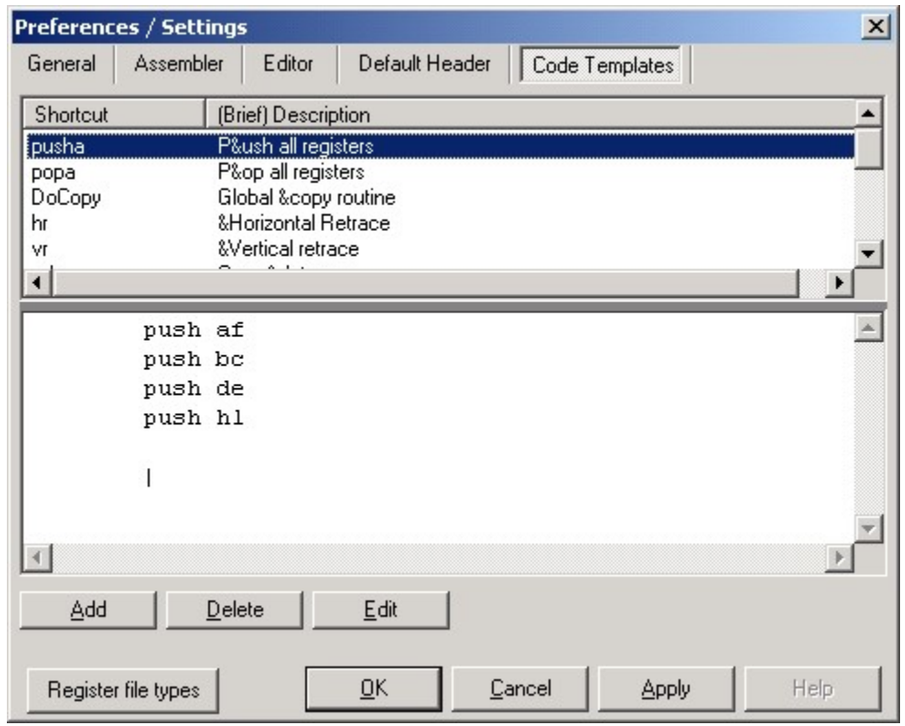

Figure 8. The code templates page of the settings screen.

What is a code template?

Code templates are one of the very powerful features of CA3. Basically a code template is nothing more than a piece of code. This code you can then insert in the program you are coding, at any place and any time, just by typing the code template shortcut and giving a call to the code template processor.
In this way you can insert 'often used' code in a few letters of typing, which decreases the release date of your project!

How do I edit these code templates?
As almost every window of CA3, this window is really easy to use... Push the **Add** button to add a code template, **Delete** to delete the selected code template and **Edit** to edit the selected code template. The edit and delete buttons are only enabled if they 'really' will do something, just as the **apply** button.

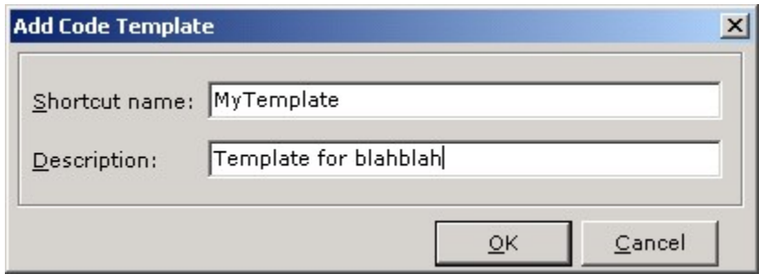

Figure 9. Add a new template

When you click the **Add** button, this screen will pop up. You can give a **shortcut name** and a description to your new code template. The shortcut is used when you want to insert a code template into your source as mentioned before in this topic. The description part is used by the main window of CA3 (in the **Template** item of the main menu, we'll get to this later)

*Insert an **ampersand** (&) in the description to let the item have a hotkey, e.g. when you have this description: **My &code template**, and you pop up the template menu in CA3, it will be displayed as **My code template**. In this way, when you press the '**C**' key, CA3 will insert the **My code template** template.*

The little editor is available when you've selected a code template in the list box. If you've done that, this edit box is always waiting for your input. You enter the actual code for your code template in this edit box. Also this edit box '**auto saves**' the template when you e.g. select another template.

Insert a pipe symbol (|) in the code of your template to indicate the cursor position after inserting the code template into your source...

Although the **edit button** seems to have something to do with the edit box, this is not true. As said, the edit box is always edit able when a code template is selected. The **edit button** let's you edit the **Shortcut name** and the **Description** of the code template that's been selected in a screen similar to the screen showed in **Figure 9**.

3.3. Main window

3.3.1. Main menu

3.3.1.1. File menu

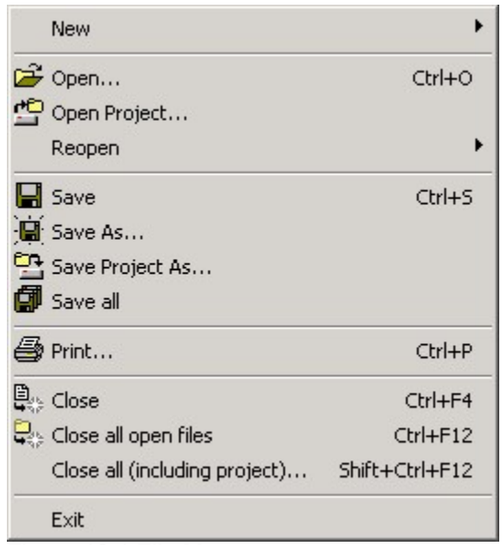

Figure 10.1 File Menu

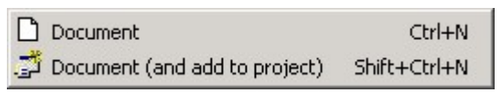

Figure 10.2 File Menu (*NEW*)

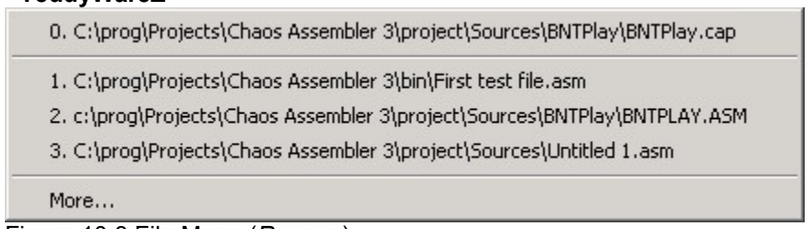

Figure 10.3 File Menu (*Reopen*)

New

When you point your mouse to new, figure 10.2 will appear.

Document
Click this to create a new source file. Directly after you clicked on this item, a new **edit child** will be created, and if enabled, the default header will be pasted at the top of the new file.

Document (and add to project)
This is the same as the other **Document** item, except for the **project**. CA3 can work with projects, this will be explained later. When you click this item a new document will be created but it will also be added to the project you are working on. If you don't have a project open, a new project will be created.

Open

This option will open the screen showed in figure 11. With this screen (which is the default windows open dialog) you can select a file to open with CA3. When you select an **.ASM** file, CA3 will open a new **edit child** and fill it with the text inside the file you've selected.
When you open a **.CAP** (Chaos Assembler Project) file, explained later, it will open a CA3 project file. As said, this will be explained later on in this documentation.

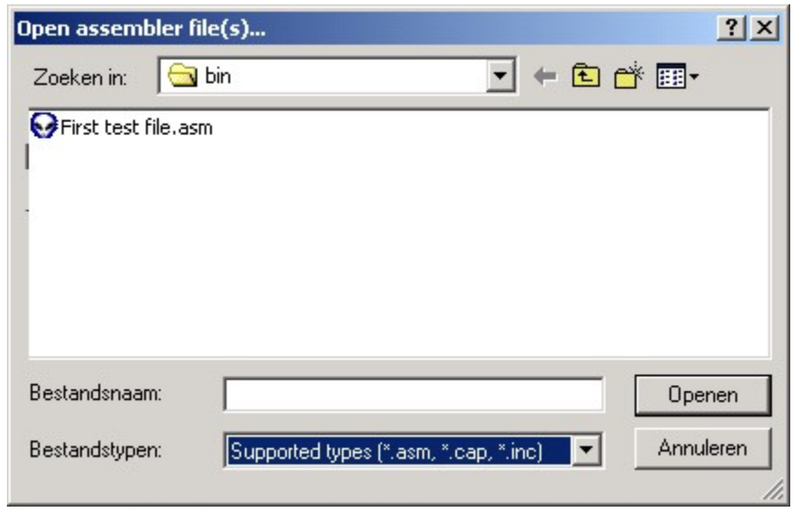

Figure 11. Open dialog

*As you might notice from this window, is that the text is **NOT ENGLISH**. Actually it's dutch. We didn't do this on purpose. Because it's a default windows dialog, it's shown in the language of the installed windows. In your case it will be in the language YOU have installed windows under.*

Open Project

See Open, the only difference is that this window only can open project files, in stead of *All supported types*.

Reopen

General
The reopen menu is used to open the most recent files with one click. In the settings screen you can edit the number of items displayed in this menu item. You see two 'delimiters'. Above the first delimiter are the most recent opened **.ASM** files. Below the first delimiter you see the most recent opened **.CAP** (Project) files.

More...
Below the second delimiter you see an item called *'More*'. When you click this item, the screen showed in figure 12 will be opened.

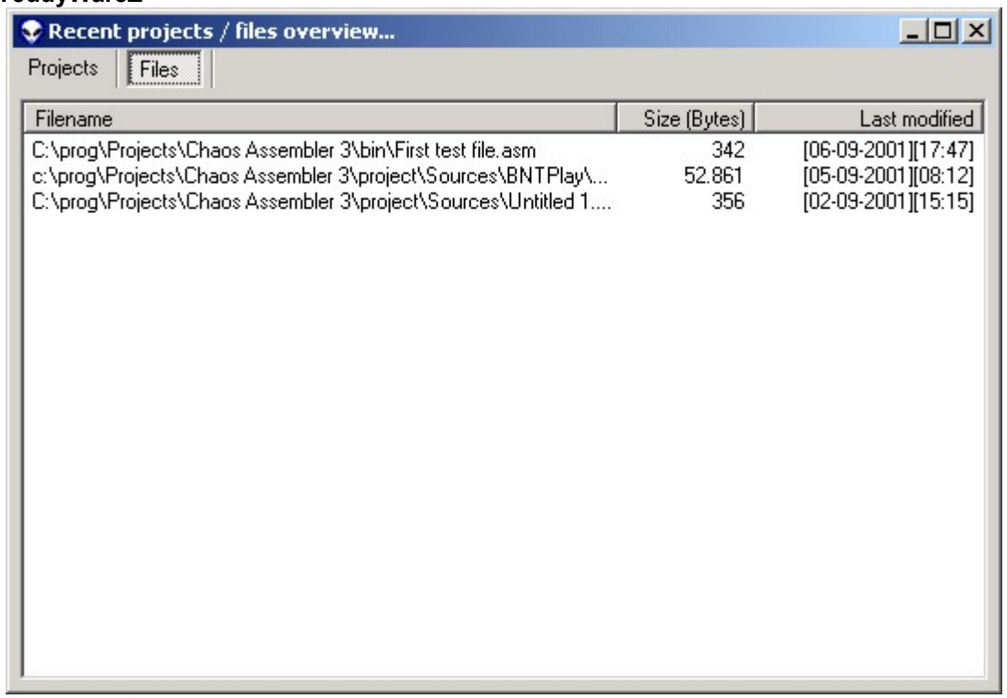

Figure 12. The open list.

In this screen you have two tab sheets. The first one will display all projects **EVER** opened by CA3. The second one will display ALL other files ever opened. If you click on one of the headings of the list (*Filename, Size (Bytes), Last Modified*), the list will be sorted on that column. Clicking on the same column, will **reverse** the sorted column. In this way you can sort in **ASCENDING** and **DESCENDING** order.

If you **Double click** on a file or project, CA3 will open the file you requested.

Save
Click this item to save the **active child edit** to disk. if the file wasn't saved before (new file) than a pop-up dialog will appear, similar to figure 11. In this dialog you can specify a filename and look for an appropriate location. If however the file was saved before or you opened the file and you click this item, the existing file will be overwritten with the current text in the **active child edit**.

Save as...
Similar to the *save* option, the only difference is, this option will always open the save dialog even if the file has been saved before.

Save project as...
This is similar to the *save as* option, the only difference is that it will not save the active edit window, but the project file (the .CAP file).

Save all
Click this item to save **ALL files opened** (edit childs) and the **project file** at once. If necessary, a save dialog will be shown (of course only for the files that haven't been saved before).

Print
Click this option to get a preview of the **active edit child**. In this screen you can see how it will look if you print the text in the edit window. In that screen you can select to actually print the text.

Close
Closes the **active child**. This doesn't have to be an edit child. This can be any kind of child window.

Close all (including project)
Closes all **active childs** and also closes the open **project**. of course this option will only be available when a **project** is open.

Exit
Close Chaos Assembler 3 (You probably don't need to use this option :D).

3.3.1.2. Edit menu

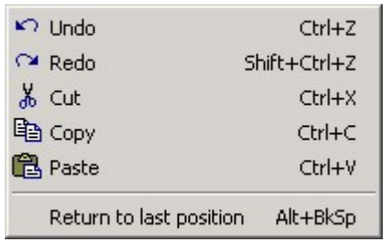

Figure 13. The edit menu.

This menu will only be available if the **active child** is an **edit child**. And even when there's an edit child opened, it may be that some of these options are not enabled.

Undo
Undo last changes made to the active **edit child**.

Redo
Redo the last undo made to the active edit child.

Cut
Move the selected text in the edit child to the windows clipboard.

Copy
Copy the selected text in the edit child to the windows clipboard.

Paste
Paste the text stored on the windows clipboard in the active edit child at the current cursor position.

Return to last position
This option will only be enabled if you've 'clicked' on a label. After you've clicked on a label, CA3 will jump to the declaration of the label and with this option you can return to the place from where you jumped to the label.

3.3.1.3. View menu

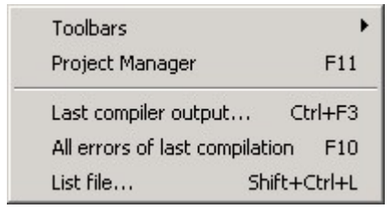

Figure 14.1 The View menu

The view menu handles the layout of the screen.

Toolbars

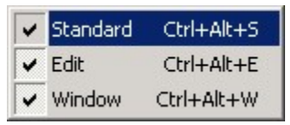

Figure 14.2 Toolbars

Click on one of these items to enable or disable a tool bar. Toolbars are the little bars below the main menu (explained further on in this document). If a tool bar is currently visible, the item of the tool bar is '*checked*'

Project manager
When you click this item, the project manager will become visible or not visible, according to the current visibility.

Last compiler output
Will open a screen with the complete compiler output of the last compiled files, as you can see in figure 15.

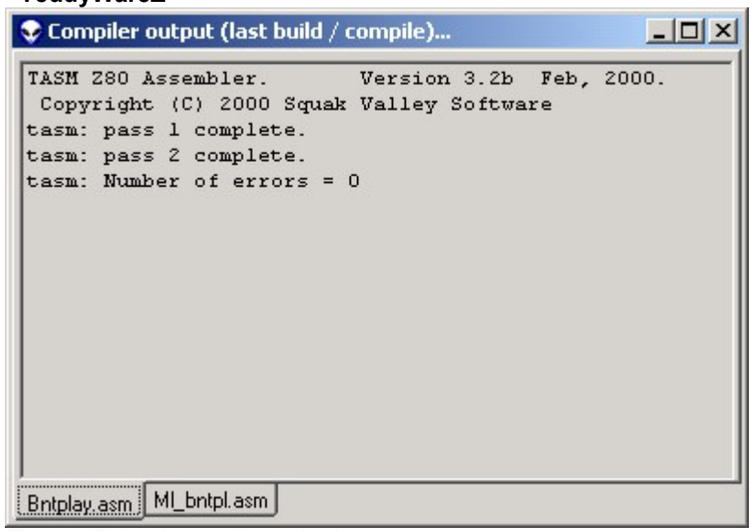

Figure 15. Last compiler output

As you can see here, this window shows the compiler output for the latest compilation. If the last compilation consisted out of more than one file, you can view the compiler output for every separate file by selecting the **tab sheet** with the correct filename at the **bottom side** of this window.

All errors of last compilation

If you compile a single file, this option is not really necessary because you can view all errors in the edit child that contains the errors, but when you've **built a project**, which can consist out of more than one file, it might come in handy to have an overview of all errors in all files. Figure 16 shows the window I'm referring to.

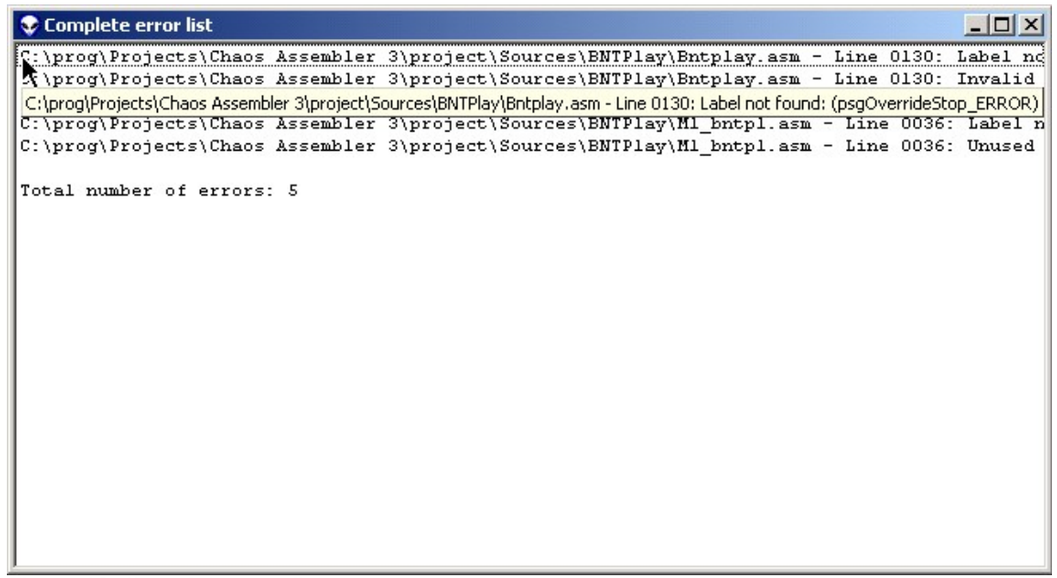

Figure 16. Complete error list

As you can see here, two files in the last compilation contained errors. Check out the file names (**Bntplay.asm** and **Ml_bntpl.asm**). Also You can see that not all of the text per line fits in the window (horizontally). To get a view of the complete line, just let the mouse **FLOAT** over the line you want to see, and you'll get a pop-up **HINT** window which will contain the complete line! Throughout CA3 there are more lists like this one. In all these lists you can pop-up such a **HINT** window!

List file...

All files compiled by TASM will produce a list file (if enabled in the settings screen). A list file is a file that contains your source and the opcodes related to your source lines.
You can only view the list file of the **active edit child**. This means the active child has to be a edit child. Also, the active edit child has to be compiled before this options becomes available.

See figure 17.

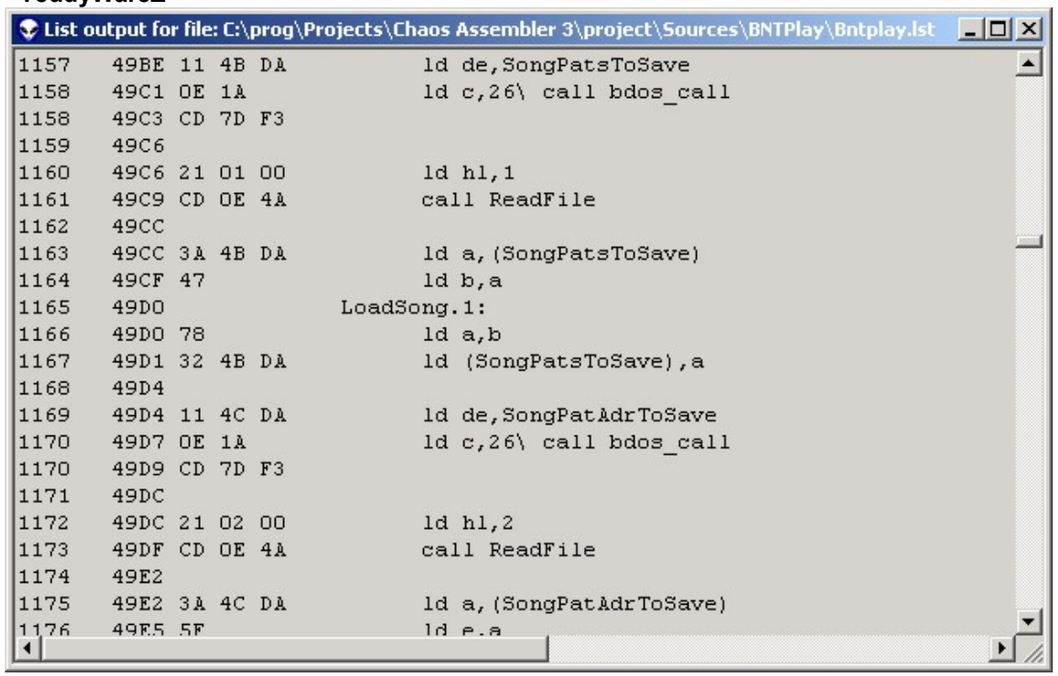

Figure 17. List output

This is the window you can view the list file of a compiled file. Not much to say about this, just notice the **MNEMONICS** right and the corresponding **OPCODES** left.
The first characters of each line are the lines as they are in the source file...

3.3.1.4. Search menu

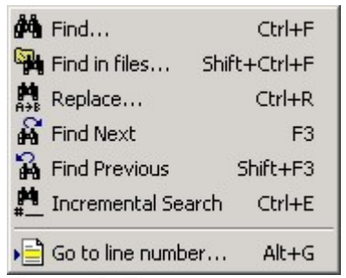

Figure 18. The search menu

*All these options are ONLY available if the **active child** is an **edit child**!*

Figure 19. The find dialog

The first item in the search menu is the **find** option. When you click that item, this dialog window will pop-up. You can enter your search command in here.

Search for
Type the string you are searching for after this caption in the **drop down edit box**. You can drop down the box to select the most recent search strings.

Case sensitivity
If checked then the search will be executed with **CASE SENSITIVITY**, this means the string you are searching for will be LITERALLY in the text, with character case in mind.

Whole words only
Check this if you don't want CA3 to search for the string in a part of a word, so if this is checked, there's assumed that your search string is a **complete** word.

Search from caret
When checked, CA3 will begin its search from of the current cursor position, in stead of starting at the top of the file.

Selected text only
If checked, CA3 will only search for the search string in the part of the file that's selected.

Direction
Forward means search from somewhere till the **end** of the file, **backward** means search from somewhere till the **beginning** of the file.

Another search feature in CA3 is the so called *search in files* search action.

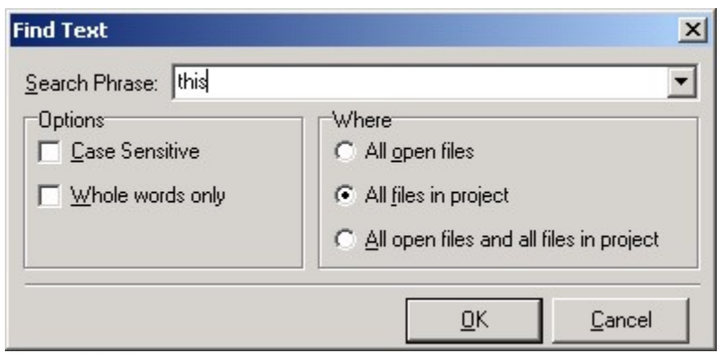

Figure 20. The find in files dialog

Here it is... The two options (Case sensitive and Whole words only) at the left are the same as in the default find dialog. The difference between this find dialog and the other one is quite big though.
The default find dialog only can search in the active edit child.

This find dialog however can search in three different ways:

All open files
When checked, the search will be committed to all open **edit childs**.

All files in project
When checked, the search will be committed to all files in the project. There won't have to be a **edit child** open for this operation, though you HAVE to have a project available, as there will be searched in all files in the project in stead of open files.

All open files and all files in project
Combine the two listed above and you know what the dialog will search for when you start searching.

When you start your search, all hits will be combined into one big list, and a window will be shown with all hits in it (see figure 21).

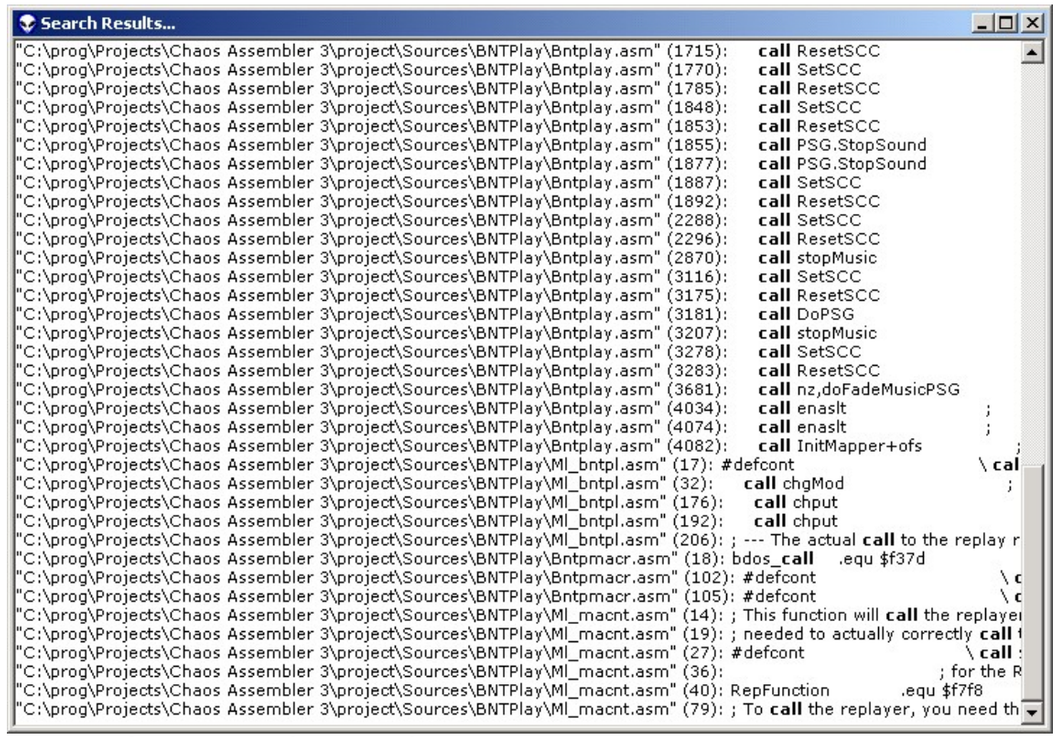

Figure 21. Search result of search in files.

As you can see here, all hits are listed in this list. Notice that everywhere you look, you see the word '**call**' in bold style. You've guessed it, that's the word where I was looking for. When you double click on a line, the corresponding file will be **focussed** or **opened** and the line where the search string has been found will be completely selected. The separate hits are divided into three

parts. First it contains the **filename** between double quotes ("). After that you see the **line** where the string was found and last you see the complete line where the string was found.

Replace...
This dialog is similar to the find dialog. The difference is that this one can replace the found text string into another. The extra options in this dialog are self explaining, so no space wasted on that issue :) .

Find next
When you've searched for a string using the find dialog or the incremental search (explained after this) you can search for the same string again by the Find next option. It will start a 'new' search from of the first character AFTER the latest hit.

Find previous
See Find next, only difference is that it will search **BACKWARDS** in the file in stead of **FORWARDS**.

Incremental search
One of the most powerful (and probably new for you) search action is the *Incremental search*. Incremental search can be interpreted as type and search at once.
When you select this option, your **edit child** will look like the one in figure 21.

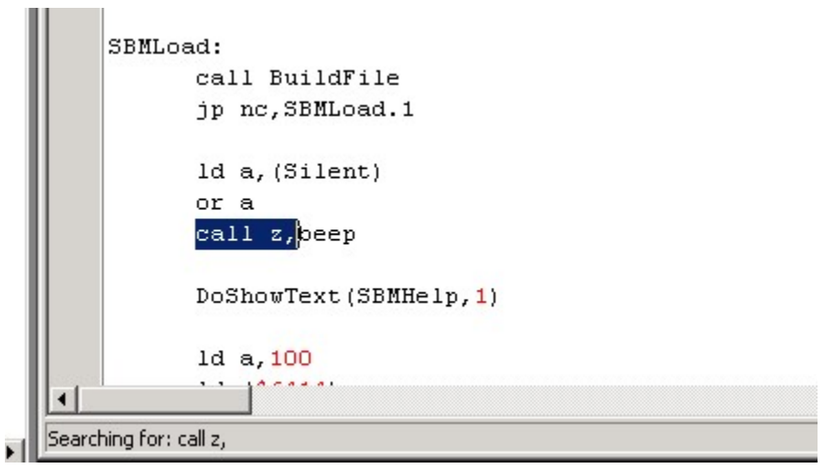

Figure 21. Incremental search in action

Here you see the incremental search in action. As you can see, the edit child has got a status bar in which you see '**Searching for:**'. The concept is really easy. When the incremental search is enabled, just type, and the selection in the file will change to what you've typed. It's kind of hard to explain, just look at the figure. I've searched for '**call z,**' and as you can see the first line found where **call z,** is in, is selected and showed.
The *Find next* and *Find previous* options also work in conjunction with the incremental search.

Go to line number...
This last search action is a quite simple one. It opens a dialog where you can enter a line number and when you press OK, the line you've entered will be selected. That's all.

3.3.1.5. Project menu

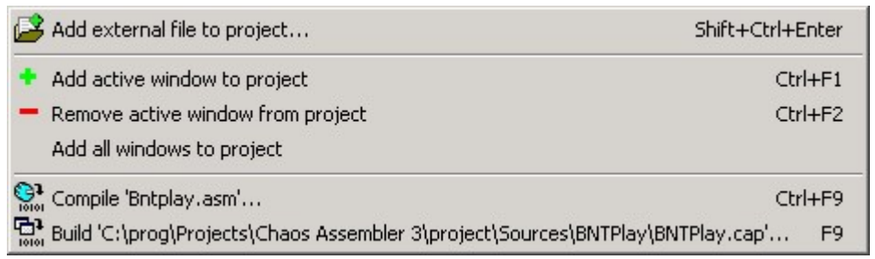

Figure 22. The project menu

Add external file to project...
This will open the already explained open dialog. You can select a file and when it's selected, CA3 will add the file to the current project. If there's no project available, a new project will be created.

Add active window to project
This will add the **active edit child** to the current project. If there's no project available, a new project will be created. If the **active child** is not an **edit child** then this option will not be enabled.

Remove active window from project
This removes the **active edit child** from the project. If the window is not part of the project,

nothing happens, if the window is the last file in the project, the project will be closed (no project will be available anymore).

Add all windows to project
This option is the same as the *Add active window to project*, the difference is that this option adds all open **edit childs** to the project.

Compile 'Filename'...
This option compiles the active **edit child**. It will open the compile window and execute TASM to compile the file. More about compiling files later on in this help file.

Compile 'Project Filename'...
This option will build the project if a project is available. It will open the compile window and compile all files in the project that need to be compiled. More about building projects later on in this help file (in the chapter where I will discuss projects).

3.3.1.6. Template menu

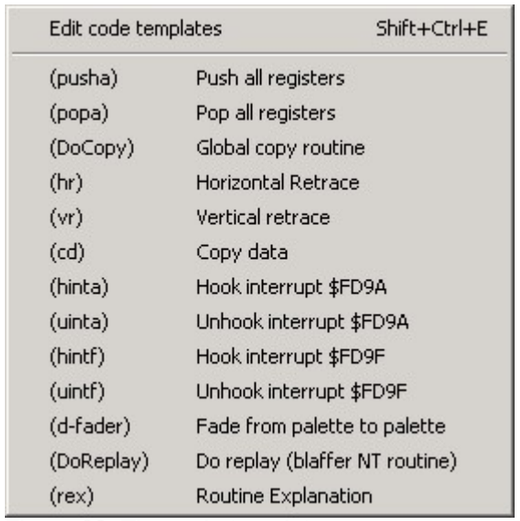

Figure 22. The code templates menu

The templates menu is not really necessary, but just added for convenience. There are three reasons why this menu is here.

The *Edit code templates* menu item warps you directly to the code templates page in the settings screen. This menu, however, has more to it. It lists **all available code templates**. This means you can see the **Shortcut** as also the **Description** of the code templates. When you click on one of the code templates, it will be directly inserted at the caret position in the **active edit child**. That's the second convenience option. The third one is (of course) if you don't remember the shortcut name complete or you don't exactly know what the template you are referring to does, you can check it all out in this menu.
However, code templates can be inserted in the **active edit child** by typing the shortcut name and hitting **CTRL+J** (More about this later on in this document).

3.3.1.7. Extra menu

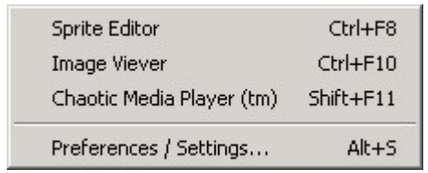

Figure 23. The extra menu

The extra menu contains the *Preferences / Settings* item which will warp you to the settings screen described earlier in this document (chapter '*The Settings Screen*').

The other two items will be discussed extensive later on in this document.

Sprite Editor
The sprite editor is a VERY POWERFUL option of CA3. It's a Sprite Mode 2 editor. And I think it's one of the first and only sprite mode 2 editors... But to save things for later, I'll quit this subject now.

Image viewer
The image viewer is another very powerful tool of CA3. With this tool you can view almost any *screen 5* and *screen 7* image formats including *palette information*!

Chaotic Media Player
The only feature of CA3 I will discuss in this topic is the Chaotic Media Player. This feature is a nice add-on to the development environment. The only thing it can do up to now is playing MP3 files. This means you won't have to use winamp anymore to play your MP3's :)
The Chaotic Media Player (CMP) has pretty much to do with the project you are working on.

When you click this option, the window showed in figure 24 will show directly. This is also a **child window**, this means it will show **inside** the **parent window**.

Figure 24. The chaotic media player!

Here you see the CMP in action! As you can see it just contains a list of files (which you can decide, of course) and some information about the current song and some buttons at the top of the window.

The buttons from left to right:

First
Selects the first song in the list.

Prior
Selects the song prior to the current song.

Next
Selects the song after the current song.

Last
Selects the last song in the list.

Play
Starts to play the selected song.

Stop
Stops the playing of the current song.

Pause
Pauses the song. Click pause again to resume.

Open song(s) and add to list
Warps you to an open dialog where you can select MP3s, these will be added to the current list.

Remove songs from list
Removes the selected song(s) from the list.

Interface
As you might think the CMP window has to be open if you want to play music, this is not true. The window is just a wrapper around the actual MP3 interface. If you close the CMP and the music is playing, the music won't stop playing, it will just continue! In this way you won't have to have the window open all the time when playing the music YOU like.
The CMP is **project dependent**. This means that the list you have in the CMP when you save a project, will be saved into the project. In this way each project can have its own CMP list.

3.3.1.8. Window menu

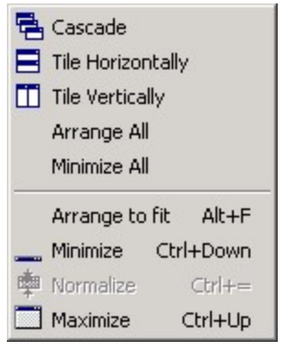

Figure 25. The window menu

Cascade, Tiles, Arrange all, Minimize all

These are default options for a program with an MDI. I'm not gonna explain this. If you don't know what these options do, please refer to a windows manual.

Arrange to fit

One of the most powerful windows options I've ever seen is the *Arrange to fit* method. It works quite simple. The concept is that you **ALWAYS** have an overview of ALL child windows you've opened. See figure 26 to see what I mean.

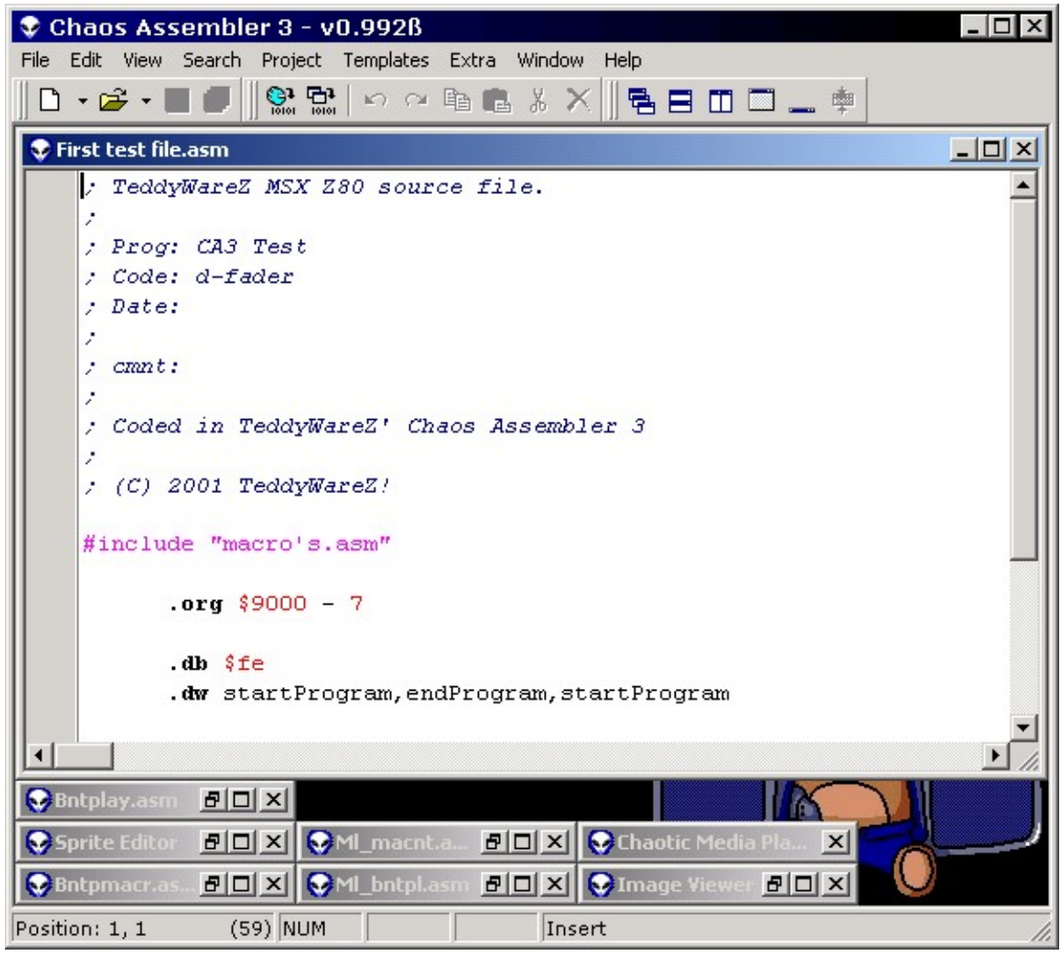

Figure 26. Arrange to fit.

The concept

Here you see the *Arrange to fit* feature in action. All windows that are minimized have the same size as the active window. This means you can ALWAYS get to ANY file you've opened. When you e.g. would double click all minimized windows, no minimized windows would be shown anymore. Executing the arrange to fit will result in the active window to fit the screen correct again and the other windows will be minimized again.
That's it, try it, I'm sure you'll love it!

Minimize

Minimizes the **active child**.

Normalize

Normalizes the **active child**, i.e. that the active child will not be minimized nor maximized, but will have the '**default**' window properties.

Maximize

Maximizes the **active child**.

3.3.2. Toolbars

3.3.2.1. Standard toolbar

Figure 27. The standard tool bar.

All toolbars in the program have their corresponding action also listed in the main menu. This standard tool bar buttons are ALL listed in the File menu of the application.

I'll discuss the buttons from left to right.

New

When you click on the little white paper, a new document will be created (but not added to the

project). This is the default action of this button. When you click the '**down arrow**' of this button, You'll get a pop-up just as explained in the **New** section at the **File menu** topic (**Main Menu** chapter).
In this way you can also create a new document and add it to the project.

Open
When you click on this button, you'll be warped to the **open** dialog as described in the **File menu** topic. If you however click the down arrow, you'll get a menu shown in figure 28.

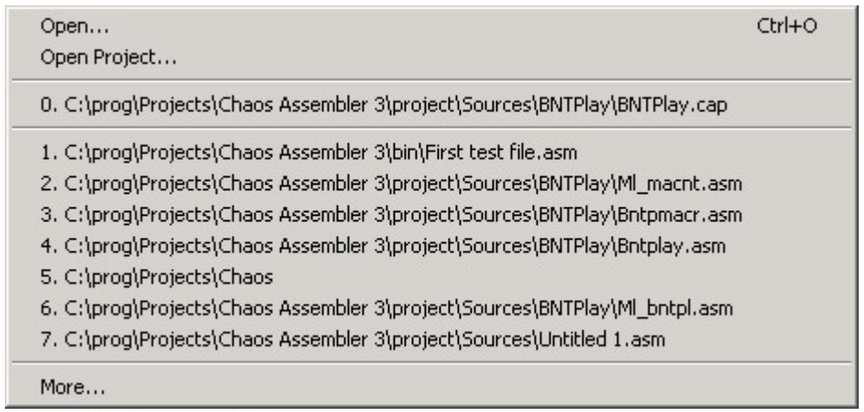

Figure 28. The open button pop-up.

When you click the **down arrow** on the **open button**, you'll get this pop-up menu. In this way you can just open a file, as the normal open, as also open a project, and the complete **Reopen** menu is stored here too. Quite handy, if you ask me.

Save
Save the **active edit child**, exactly the same as '**File > Save**'.

Save as...
Save the **active edit child** as another file, exactly the same as '**File > Save as...**'.

3.3.2.2. Edit toolbar


Figure 29. The edit tool bar

All toolbars in the program have their corresponding action also listed in the main menu. This standard tool bar buttons are ALL listed in the File menu of the application.

I'll discuss the buttons from left to right.

Compile
Compiles active **edit child** see also **Project > Compile 'Filename'**.

Build project
Builds the active **project** see also **Project > Build 'Project Filename'**.

Undo
Undo last changes, see also **Edit > Undo**.

Redo
Redo last undo, see also **Edit > Redo**.

Copy
Copies selected text to the windows clipboard, see also **Edit > Copy**.

Paste
Pastes test saved on the windows clipboard into the active edit child at the current caret position, see also **Edit > Paste**.

Cut
Moves selected text to the windows clipboard, see also **Edit > Cut**.

Delete
Deletes selected text, see also **Edit > Delete**.

3.3.2.3. Window toolbar

Figure 30. The window tool bar

All toolbars in the program have their corresponding action also listed in the main menu. This standard tool bar buttons are ALL listed in the File menu of the application.

I'll discuss the buttons from left to right.

Cascade
Cascades all active windows (this means not minimized windows). Default MDI application option.

Tile vertically
Tiles all active windows (this means not minimized windows) vertically. Default MDI application option.

Tile Horizontally
Tiles all active windows (this means not minimized windows) horizontally. Default MDI application option.

Maximize
Maximizes the **active child**.

Minimize
Minimizes the **active child**.

Normalize
Normalizes the **active child**, i.e. that the active child will not be minimized nor maximized, but will have the '**default**' window properties.

3.4. Projects

3.4.1. What is a project?

Up until now there was no project support in any of the assemblers I've ever seen made on / for MSX. CA3 will change all of this. From of now you will almost be FORCED to use projects... It's one of the features every assembler should have, but none has. As said, CA3 supports projects.

A project actually is nothing more than a number of files grabbed together. A good example is Blaffer NT. Blaffer NT was made with CA2 which doesn't support projects. Blaffer NT consists out of approximately 15 ASM files. When programming the product, I had to have all files open at once or open them as I needed them. This kind of sucks, especially when you show your source to other users, they won't understand which files belong together and that makes the readability fairly small...
Chaos Assembler 3 is proud to introduce project support. This means you will have a list of files which you can open with a single click. Compiling files you don't have open? No problem, the project manager can do it...

Another neat feature of the project manager is that it can make **destination files**. Normally the compiler will output the binaries it made into the same directory as the source file, with the same name as the source file and the '**.obj**' extension. CA3 let's you decide where to put the compiled file and also with WHAT name! This means you can save your ASM files onto the disk in the **disk drive**, and that disk you can then run directly on your MSX!

3.4.2. CA3 and projects

of course (probably) directly after reading the introduction topic of this chapter you wanted to know how exactly projects work in CA3. Well, this chapter will discuss the complete implementation of projects in CA3.
A project is something under water in CA3, just like the Chaotic Media Player (Mentioned earlier). You don't have to see anything of a project whilst it still can be there. In stead, CA3 has got a **project manager**, with which you can manage your projects. So the **project manager** actually is a wrapper around the project support in CA3.

Projects are saved as **.cap** files. Do not edit these files manually. Let CA3 handle these files for you, else you will probably mess up your project!
So far for the warnings, up to what is saved in a project file.

First of all, a project file saves filenames. All file locations + names that are in a project will be saved into the .cap file. Note that it **DOES NOT** save the complete files into the cap file, only the filenames so those files can be reopened after you've opened the project again. One **project file** consists out of two filenames and a property. The first filename is the **source file** you are editing. The second filename is the **destination file**. This means the file where the compiled source will be saved to.
Last a project file contains a **true** or **false** property, which indicates whether the file should be compiled during the **build** of a project. More about building projects later.

Secondly, a project file contains all kinds of window positions. It will remember ALL of your windows and there positions. This means if you had 2 **edit childs**, an **image viewer** and the **project manager** window open whilst saving the project file, CA3 will reopen the edit childs, image viewer and the project manager at the same place and with the same size as they were when you saved the project! The project manager will EVEN save the list of the Chaotic Media PLayer at that moment (if there is a CMP list available). And yes. Even when you were playing a song when you saved the project, CA3 will start playing THAT song again after opening the project.

As an example, I will use the BNTPlay.cap project. This example project is delivered with CA3.

3.4.3. The project manager

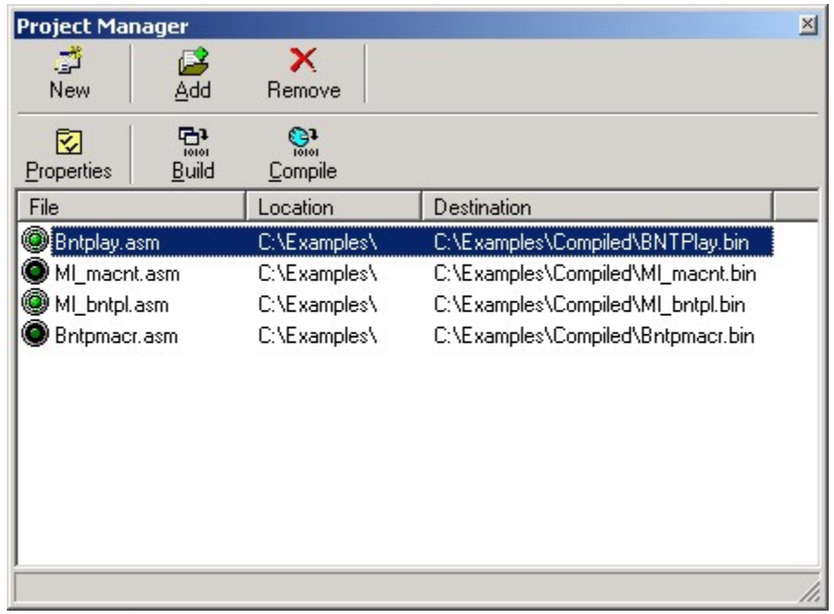

Figure 31. The project manager

Here you see a small window called the **project manager**. There really isn't much to say about this window, cause it's small and really easy to use.

Some things you should know

- Something quite important is the moving of project files. This has no function at all, but is for the readability. When you press **right mouse button** on the **list box** you'll get a menu, with which you can MOVE one project file **up** or **down**. In this way you can keep your files together, just as this project has done.
- When you add a file to a project or save a new file for the first time that is in a project, the destination file will be the located in the same directory as the source file with the **.bin** extension.
- Project files will be saved **relational** if that is possible. That means that if the project file (**.cap**) is located in the directory **c:\examples\BNTPlay.cap** and the source files are located in **c:\examples\source*.asm** (e.g. **c:\examples\source\BNTPlay.asm**) and you would copy the complete example directory to **c:\prog\projects\CA3\examples** there will be assumed that the **BNTPlay.asm** file is located in the **c:\prog\projects\CA3\examples\source** directory. However, if the file could not be relational saved (e.g. .cap file is located in **c:\examples** and

the source file is located in **c:\prog\default files**, the filename will be saved **absolute**. If you save a file in the **default project directory** (see settings), it will be saved relational in a different way. This means if the **default project directory** changes, there is assumed this file is also located in that new directory. This can come in handy when you're using sources on another computer with a different defaults project directory but with the file you're referring to in that directory!

The tool buttons

(the big buttons at the top of the window).

New

This button corresponds to the main windows '**File > New > Document and add to project**'. It creates a new file, will open it and adds it to the project.

Add

This button corresponds to the main windows '**Project > Add external file to project**'. It opens an open dialog and if you select a file there it will be added to the project (so it will be directly visible in the list you see).

Remove

Removes the selected file (in the list) from the project. If it's the last file in the project, the project will be closed.

Properties

This will open the properties window of the selected file. In that window (shown in figure 32) you can select the **destination file** and whether the file should be **compiled** during the **build** of the project. More about this you can find at the CA3 and projects and the last paragraph of the UNREGISTERED DEMO VERSION sections.

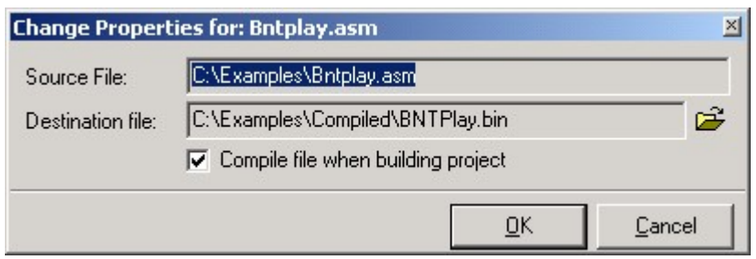

Figure 33. Properties window for a project file

The source file is not edit able. That's because that's the identification of a project file. You can change it by using another file to add to the project.

Destination file

Normally TASM (the compiler) will output the binaries it made into the same directory as the source file, with the same name as the source file and the '**.obj**' extension. CA3 let's you decide where to put the compiled file and also with WHAT name! This means you can save your ASM files onto the disk in the **disk drive**, and that disk you can then run directly on your MSX!
Click on the little folder right to the destination file to select the file. It will try to open a save dialog where you can specify a file. The file you select does not have to exist in order to let it be saved to that file, and even when you select e.g. **a:\myfile.bin** and there's no disk in drive a, CA3 will still allow you to save to that file!

Compile file when building project

As said, a project in CA3 is just a list of files that need to be compiled. But as not every file in a project has to be compiled (e.g. because you have some macro's listed in another file, which is included in your source) you can instruct CA3 to compile the file you've selected or not. This is only used for **building** projects. When you build a project, every file that has the *Compile file when building project* option enabled will be compiled. You can see if a file has to be compiled in the list below the buttons. If the green light is **ON** the file will be compiled during a build, if **OFF** it will not be compiled.
In the example project you see **four** files from which two the LED is **ON**. These files are actual programs that you run on MSX. The two files with the LED **OFF** are included in the corresponding program file (The first two are together and the latter 2 are together).
These files are not compiled when this project is build, because they are used in the actual program file. (see figure 34).

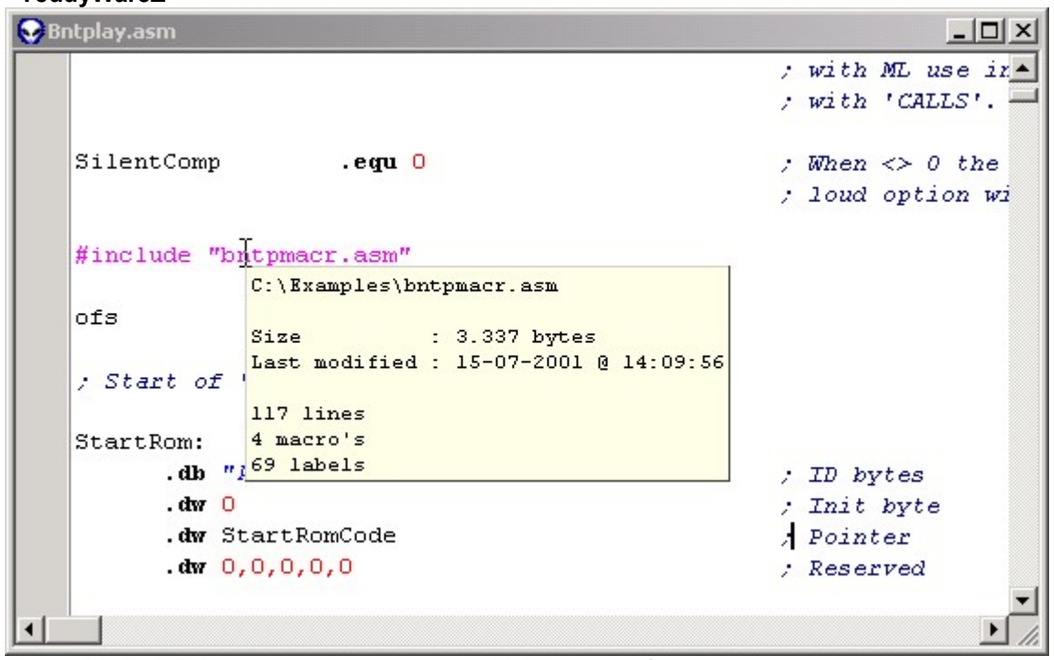

Figure 34. BNTPMacr.asm is included in the BNTPlay.asm file

Here you see a small block of code in the *BNTPlay.asm* file. As you can see (looking at the #include directive) *BNTPMacr.asm* is included in the BNTPlay.asm file. This means that the BNTPlay.asm file can call macro's defined in the BNTPMacr.asm file. If this directive wasn't there, the file would not compile cause it DOES make calls to macro's and labels defined in the macro file.
Also you can see **ONE** of the many great aspects of the **edit child**. You see a little pop-up window with all kinds of information of the BNTPMacr.asm file. When you **float** the mouse over a filename at an include directive you'll get this information after just a really short while waiting! Just by seeing this, you can tell if the file is the right file (size, date / time) and you also can see how many **lines** the file contains, how many **macro's** are defined and how many **labels** the file has... If you would press **CTRL** whilst floating over the file, the cursor would change into a little hand, a click would then result in CA3 opening the file!
Great, great, great, now isn't it? More about the **edit child** later. This was just to let you see the BNTPMacr.asm file does not have to be compiled, as it is included in the BNTPlay.asm file!

Building projects and compiling files
In the next chapter I will handle the compiling and building of files.

3.5. Compiling files and Building projects

3.5.1. The concept

When you use CA3, you use it to create MSX assembly programs / products. That means you will have to have binaries that are runnable on MSX. CA3 uses TASM to compile the sources you've created on MSX (see also at the Introduction section of this help file).
There are two ways of compiling. The first one is the compiling of one file. The second one is the building of projects. The difference between these two is that if you build a project, you just compile all the files in the project (that need to be compiled that is) after each other. Building projects should come in handy when you want to make sure you have the newest compiled binaries of every file. And it just is a lot easier than compiling each file separately.

CA3 has a really fancy looking compile window which will tell you what's happening during the compilation of one or more files.

3.5.2. Compiling files

Of compiling and building projects, you will probably use the compile option (for one file) the most. Because you always work on one file at a time, this option is there to compile just that file. There are two ways of compiling the file you are working on. First of all is via the **Compile** button of the project manager. If you press that button, the file you've selected will be saved if that's necessary, and after that a nice compilation window will appear. This window is the window where you can see the progress of the compiled file and whether it was compiled correctly (as shown in figure 35). The second way is via the **main menu**. The option **Project > Compile 'filename.asm'** will compile the **active edit child**. This option is also available on the tool bar and can also be executed by hitting **CTRL+F9** in the active edit child... This is quite easy, whilst you are editing, you can compile real fast by just hitting **CTRL+F9**.

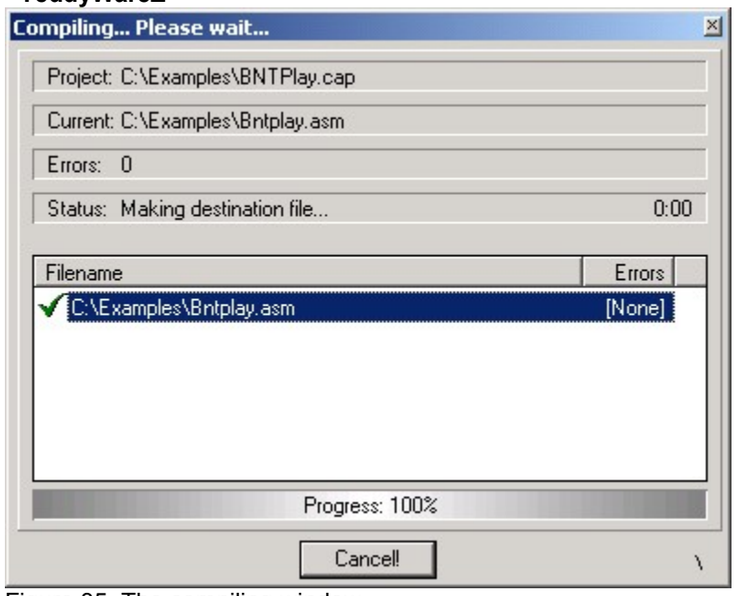

Figure 35. The compiling window

Here you see the compile window in action. I grabbed the image as fast as I could, but I couldn't prevent the file to already be compiled. In the state you see CA3 is making the destination file. This already shows the power of TASM. This source file was **4192** lines of code! CA3 first compiles the file and after that it will create the destination file. If the file you compiled did not belong to the project you are working on or you haven't got a project open, CA3 will skip the destination file. The Destination file will be made by TASM itself in the same directory as the source file with the same name, but with the **.obj** extension.

There's quite much to see on the screen. Well, the first four text lines should be quite easy to understand. The list box contains all files which will be compiled in this compilation. The little icon next to it will tell the status. A little 'V' means 'OK!', a little '?' means that CA3 still needs to compile this file and a 'X' means that file contained errors...

If you compile a single file like this, the window will disappear as soon as the file is compiled. That means that if you compile a file using the project manager and the file you want to compile is not open at that time, this will mean that you can't see what happened and if the file was compiled correctly...
I've found something to this. If the file is compiled or project is built correctly, the title bar of CA3 will get a **green color**. If the compilation fails, the title bar will get a **red color**. If the source is compiled correctly but creating the destination file failed, the title bar will also get a **green color**.

If you **have the file open** when compiling or **open the file after the compilation** and the **compilation failed**, you will notice a list under the edit able region of the **edit child** with the corresponding source (see figure 36).

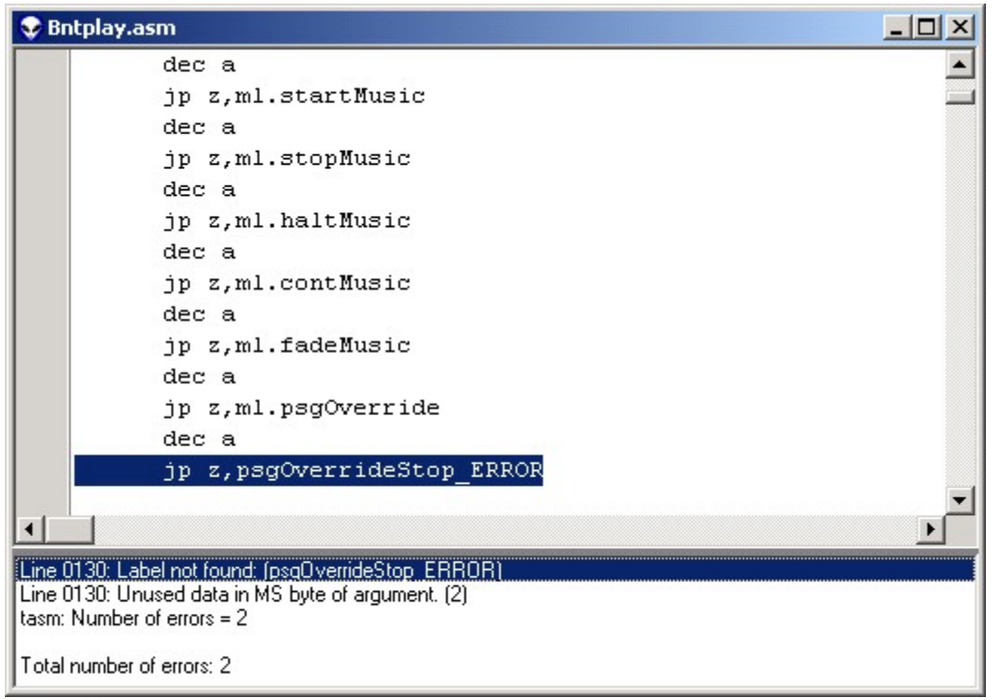

Figure 36. Compiling failed!

Here you see a compilation that failed. I'm not going to explain what that list box exactly does. That will be discussed later. What I do want to tell is that you might notice that you twice see there are 2 errors in total. This is done on purpose. First you see the number of errors TASM had. The Total number of errors count can be higher, because of the fact that creating the destination file can also result in an error.

3.5.3. Building projects

Building projects is quite the same as compiling just a file. The only difference is that you can

compile more files at once. This means relaxation for you! You can compile more files at once by just ONE single click! Or by selecting it in the main menu under **Project > Build 'Project filename'**, clicking the **build button** on the **tool bar** or just by **hitting F9**.

Building projects is as you can see really easy. In figure 37 you can see me building the complete **SCC-Blaffer NT** tracker (at least the binaries). I've chosen to show Blaffer NT here because it just has some more binaries than just the replayer. I can show the building of projects better with this project. I was quite astonished myself too, in CA2 I had to search the right file and hit the compile button and that for every file. This project has about 15 files with 5 of them to be compiled. And with just one key (**F9**) I rebuilt the COMPLETE product (at least the binaries).

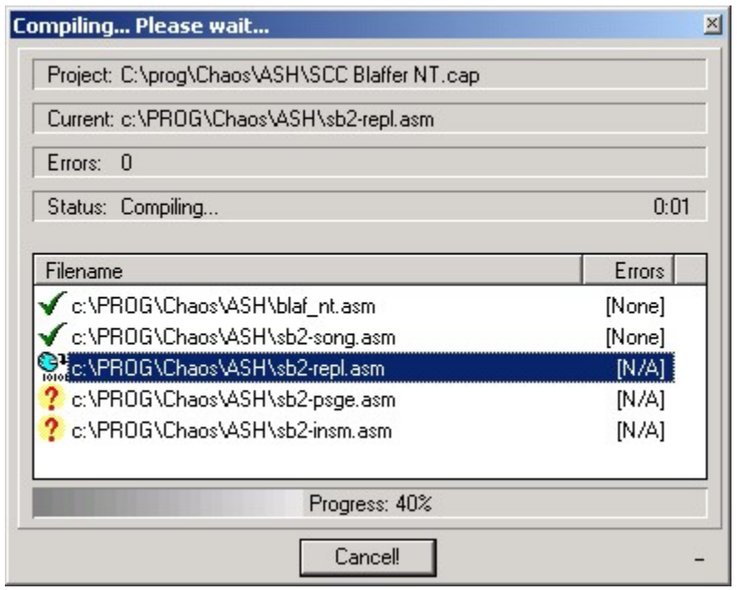

Figure 37. Building a project

As you can see here, CA3 is busy building the active project. It goes SO fast that only one second elapsed and already 2 files are compiled! OKay, you see three different icons. The first two indicate that the file is compiled and it also successfully is compiled. You can also see that at the **error** column. You can see no errors occurred due to the **[none]** caption.
The third file is the file that is being compiled at the moment. You can see that by looking at the **blue bar** and the '**compile**' icon...
The fourth and fifth file are still waiting to be compiled. You can see that by looking at the icon which has a question mark (?) in it. The three **[N/A]** captions indicate that the error count is not yet available. Only for files that have been compiled completely the error count will be available (as you can see by looking at the figure).

Another difference of **building a project** is that when one or more files that were compiled had errors, the window will not close automatically. This is done to give you an overview which files had how many errors (again: *Readability*). Figure 38 shows that.

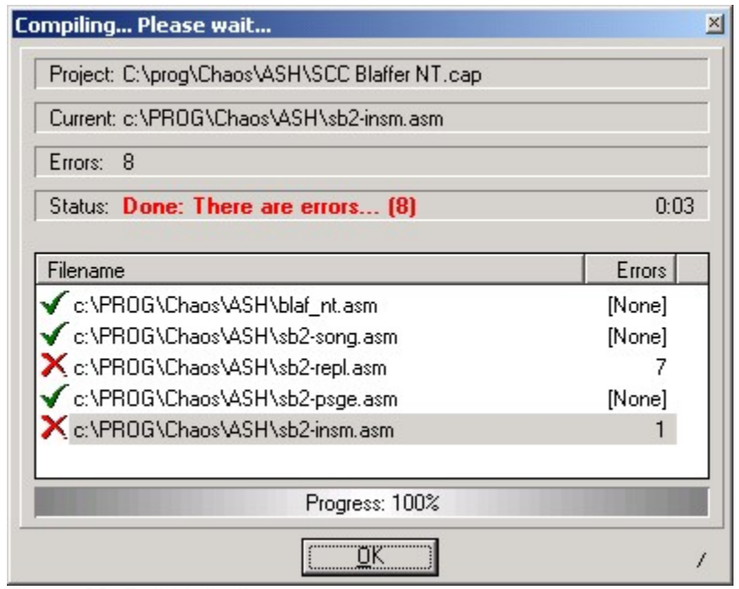

Figure 38. Build failed!

As you can see here, the build failed. I've created some errors in the two source files. In totality I've created 8 errors as you can see in the window. You also might notice the '**Cancel**' button has changed to an '**OK**' button. This is because the window doesn't close automatically, as was the story when building a project succeeds or when you compile just one file.

Click '**OK**' to close this window. After that there are enough possibilities to view the errors per file or altogether, as shown in the view menu topic of the main menu chapter.

3.6. Extra's

3.6.1. The image viewer

Chaos Assembler 3 has a built in **image viewer** with which you can view all kinds of MSX images. The image viewer not only can view the images, it also can do a lot of neat other things. You can see the image viewer in figure 38.

*Note: If the image editor is OPEN and ACTIVE, the main menu will have a NEW item: **Image viewer**. It'll appear before the **Extra** item! This menu item has all the tool buttons in it.*

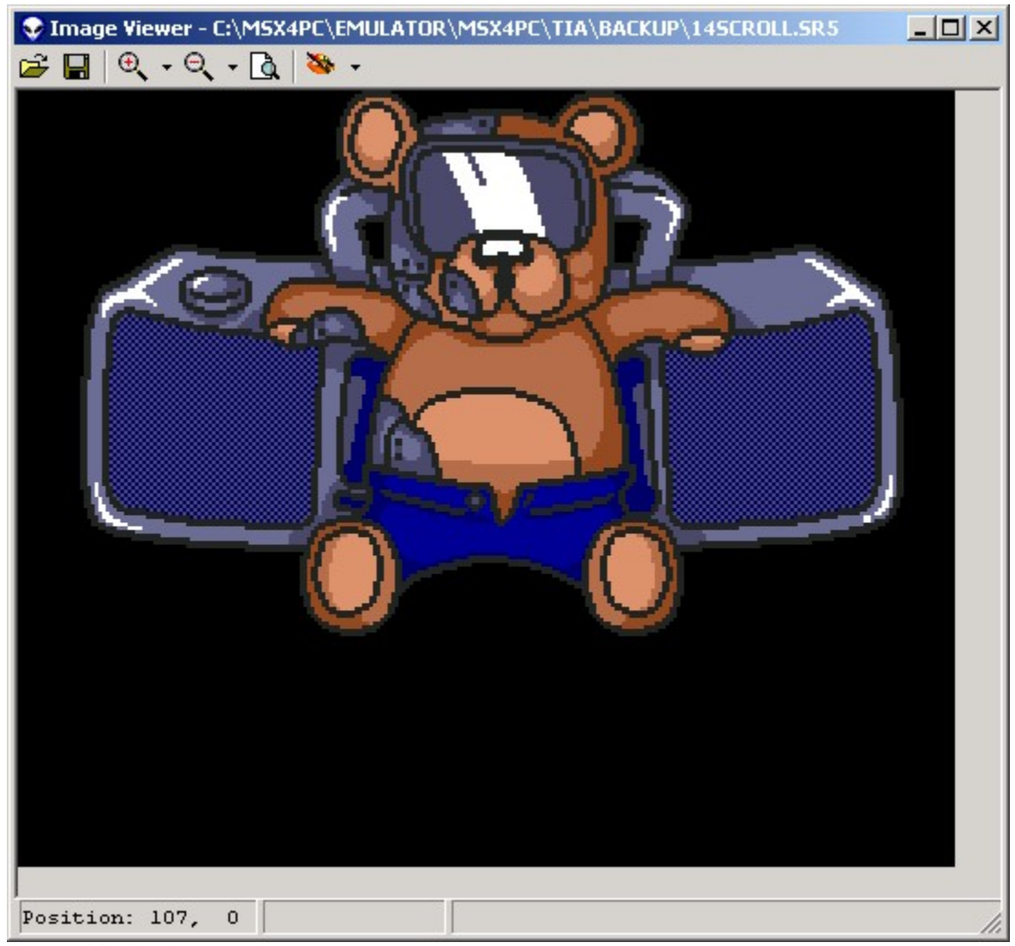

Figure 38. The image viewer

As you can see it's a window which is about 90% filled with a box where you can display an MSX image (as you can see in figure 38, the teddybear is an (Graph Saurus) .SR5 file!).
In the **title bar** of the window you can see the **filename** (with complete path) of the currently loaded image. Below that you see **6 tool buttons**. From left to right:

Open image
This button will warp you to an open dialog where you can select an MSX image to load.

- TeddyWareZ -
Currently supported types:

- **Graph Saurus Screen 5 BLOAD files (*.SR5)**
- **Graph Saurus Screen 5 COPY files (*.GL5)**
- **Unknown file format screen 5 (*.CC5)**
- **AGE (DD-GRAPH) files (*.GE5)**
- **Graph Saurus Screen 7 BLOAD files (*.SR7)**
- **Graph Saurus Screen 7 COPY files (*.GL7)**
- **Unknown file format screen 7 (*.CC7)**

The image viewer will automatically try to find a palette file (except for GE5 files) and load it. As Graph Saurus and the Unknown image format both produce 8 palettes in a single file, the image viewer needed support for that. Luckily that's built in! You can view your images in CA3 as if they were being showed on MSX! More about this later in the palette part.

Export image
Sometimes it can come in handy to export an MSX image to a PC format image. The image exporter is for that. You can export your image to a **.bmp** (windows bitmap) file. There are enough tools available to convert that image to e.g. a **.jpg** image which will be perfect for web deployment.

Zoom in
Often it comes in really handy to zoom in on the current image. That's what this option is here for. **Notice** the little **down arrow**. With this you can quick zoom.

Zoom out
Zooms out on the image (see ***Zoom in***)

Zoom default
Zooms to the default image size (i.e. x1)

Copy palette info to clipboard
This is without doubt one of the most (if not THE most) powerful options of the image viewer. Remember the times you have made an image and got a paper somewhere and you were writing down the palette information? **These days are over**! This button copies the palette information into a pasteable block of data for your sources. **You will NOT have to write ANY palette down EVER**!

You also will see a little **drop down arrow** to the right of the palette icon. Click this and you'll get the drop down menu shown in figure 39.

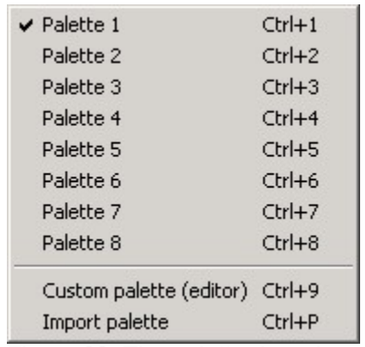

Figure 39. The palette drop down menu.

If you've loaded a **full palette** (i.e. **NOT** a .**GE5 file** or an image without a palette file) you will be able to click any of the 8 palettes to switch to that palette. If you've loaded a **.GE5**, these items just will **not** be enabled.

The **import palette** item just opens another open dialog where you can select another palette file. If you want to edit a palette, CA3 is the right tool for you. Click on the **Custom palette (editor)** to be warped to the **palette editor**, which will be described in the **next topic**!

More extras of the image viewer
Cursor position
Also a very simple but quite powerful option of the image viewer is the so called cursor location. Look at the **status bar** on the bottom of the dialog window, and move your mouse over the image. The image viewer will tell you exactly at what pixel you are moving now!

Block mode
Okay, we've discussed the palette stuff. What else is a standard action you ALWAYS take when making a demo? Yes! Writing down coordinates of objects (like fonts and equalizers). And YES, also THIS will be history from now on!
The image viewer can select blocks and tell you exactly **how big** it is, where it **starts** and where it **ends**... Try moving the mouse to a **certain position** in the image and hold the **shift** key. Now move your mouse again and see what happens! Also look at the status bar (right side). Now THAT is what I call **fitting the programmers needs**! (see figure 40).

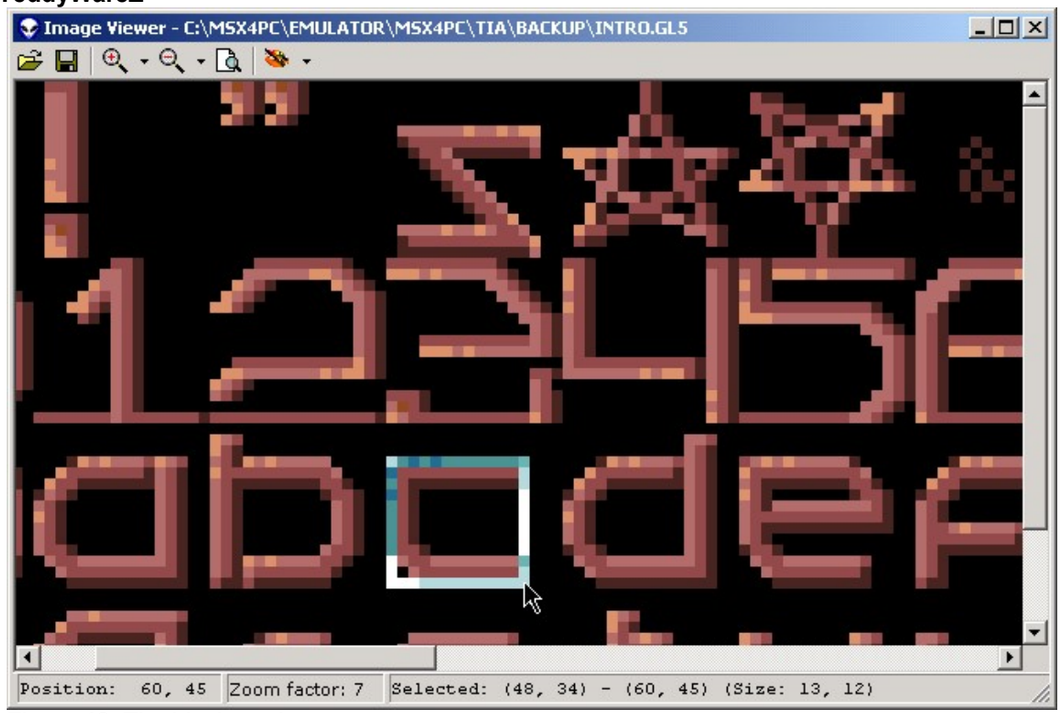

Figure 40. The block mode.

Here you see the block mode of the image viewer in action. As you can see you can select a certain block in the image to view some coordinates which can be used in your source pretty easily! Notice the block has a strange color. This is done to **always** see the block. A selected pixel has the inverted color of the original. So by holding **shift** and moving the mouse you can select these kind of blocks. Notice the **status bar** where all information is.

3.6.2. The palette editor

Another **neat** feature of the image viewer is the **palette editor**. The palette editor can be used to create your OWN palette. To get to this palette editor, drop down the arrow of the **palette** tool button on the **tool bar** of the **image viewer** and select **'Custom palette (palette editor)'**. If you click this item, you'll be warped to the dialog shown in figure 41. The palette editor can also be called by the main menu of CA3 (**Image viewer** item).

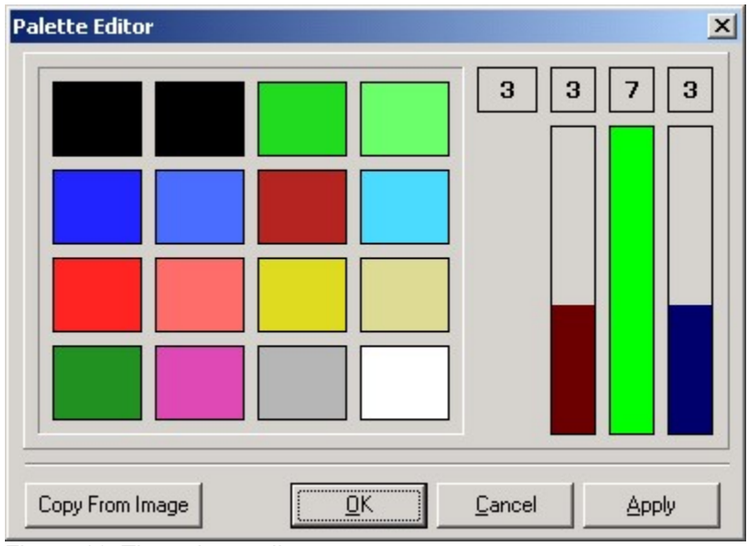

Figure 41. The palette editor

Here you see the oh so great palette editor. It's really simple. The 16 **boxes** you see are the current colors (as MSX has 16 colors). Move the mouse over one of them and the cursor will turn into a **hand point** cursor. When you click it, the current index (the color you will be editing) will be set to THAT color. You can verify that by looking at the **most left** box of the four little boxes in the upper right corner of the dialog window. To change the **red**, **green** and **blue** intensity of the color, **left** or **right click** on one of the three most right boxes (which show the current intensity of that specific color). **Left** to **increase** the intensity and **Right** to **decrease** it. If the intensity would exceed it bounds (bounds are 0 through 7), it will be clipped to the (respectively) highest (7) or lowest (0) intensity.

Copy from image
Copies the palette data from the current image. In this way you can easily change the palette according to the original palette. The palette editor will always start with the **default MSX palette**.

OK
Applies changes and closes the dialog (see ***Apply***)

Cancel
Closes the window without saving the palette information.

Apply
Sets the image palette to the palette you just edited.

3.6.3. The sprite editor

On MSX sprites are often used in games and demo's. In games they're really a must and in some demo's you see some pretty impressive sprites come by. Making sprites however is quite hard, as you'd have to make them with some little editor probably written in BASIC and what if you would have more colors in the sprite or even more colors on one line? This frankly is quite hard to make, because there are no good editors for that and thus people probably go to their graphic editor, make a sprite and from there of write down all colors and patterns. But if you have more than one color on a specific line, you'd have to split up the sprite in two sprites etc. etc. All in all, making a COOL looking sprite on MSX is quite hard.
These days are over. Chaos Assembler is proud to present (probably) the greatest (if not, the only) **sprite mode 2** editor! With CA3 you can create beautiful sprites! More colors on one line are no problem. Copying the sprite data into Z80 data structures is NO PROBLEM. That sprite data is ready to be used in your applications!
Up to the sprite editor then. In figure 41 you see the sprite editor.

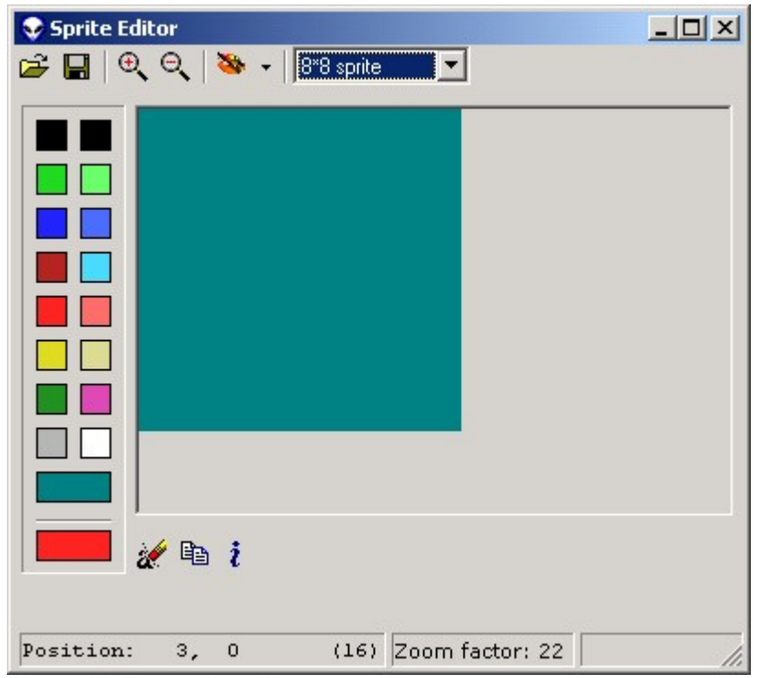

Figure 41. The sprite editor

The sprite editor also has a **main menu** extension in the **main window** of CA3, just like the image viewer has. You can find it when the sprite editor is active and it's called '**Sprite editor**'...

The sprite editor is quite easy to use. The **open** button (folder icon) let's you open an MSX image as a sprite. It'll take the first 8*8 pixels of the image.
The **save** button is there to export the sprite to a bitmap image. Then there's the **zoom buttons** again and the **palette button** (see image viewer).

Sprite mode selection
You can decide to create an 8*8 or a 16*16 sprite mode using this drop down box. If you select another mode, the sprite you have at that moment will be preserved as much as possible. That means that if you switch from 16*16 mode to 8*8 mode a quarter of the sprite will be preserved and vice versa the complete sprite will be preserved.

Color selection
You can see 17 boxes at the left side of the screen. The first sixteen represent the colors on MSX. The seventeenth color (a bit **green**) is not actually a color. It's more a **virtual color**. It represents the transparency of the sprite. A sprite is build up out of pixels. And the **pattern** says there's a pixel, or there's not one. This color is for that. When you see this color in a sprite, it'll

mean that on MSX this pixel will not be seen and thus have a transparent color.

Below those 17 boxes you see another box. This box shows the currently selected color.

To **select a color**, click on one of the color boxes. The current color immediately becomes this color. You can also change the intensity of a color. By clicking right, a little window will pop up in which you can change the intensity of the color, as shown in figure 42.

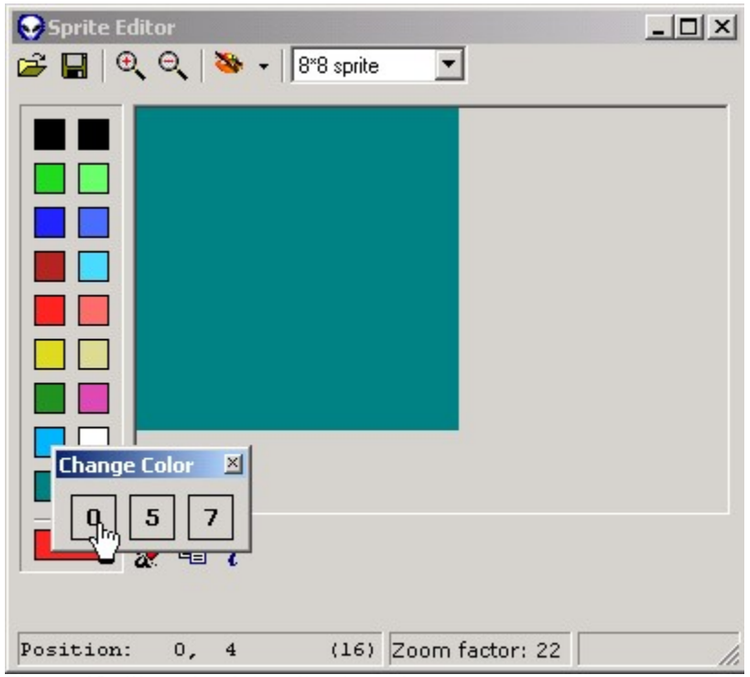

Figure 42. Changing intensity of a color

Here you see I've changed the fifteenth color of the palette from **gray** to **blue** using the color change dialog. The three boxes represent respectively the **red**, **green** and **blue** intensity of the color. By **clicking left** or **right** on them you'll change the intensity of that index.

Making a sprite
With this knowledge, it's quite easy to make a sprite. The big green part of the window you see in figures 41 and 42 is the actual sprite. Clicking on it will change the color of the selected pixel (where the mouse floats over) to the current selected color. By clicking right on any specific pixel will result in the current index to be changed to the color that pixel has. You can put as many colors in the sprite as you like, but remember, this will also increase the sprite count the sprite would take create it on the MSX...

It's **THAT** simple. More about this I can't tell you...

The other tool buttons
In figure 43 you can see I made a little sprite myself.

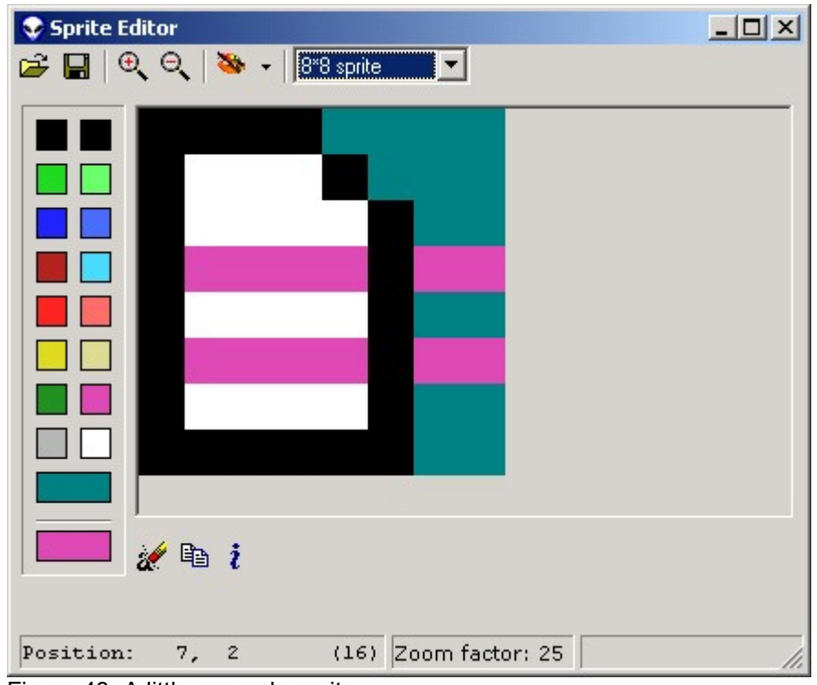

Figure 43. A little example sprite.

Here you see I've created a sprite. I guess you'll see this sprite can't be made using one MSX sprite. By using the little button with the '**i**' caption, CA3 will calculate how many sprites this on MSX will take in **sprite mode 2**. By looking at this sprite you see three colors (**black**, **white** and **purple**).
And the first thing you probably think is that this sprite will take three sprites on MSX. But this is not true. This sprite can be made using TWO MSX sprite mode 2 sprite.

As every line can have it's own color, the **first sprite** will contain all **black pixels**. The second sprite will have the **second** and **third** line filled with **white**, the **fourth** with **purple**, the **fifth** with

white again, the **sixth** will be **purple** and the **seventh** will be **white**... In that way only two sprites will be necessary! And pressing the '**i**' button will result in CA3 showing the sprite will take two sprites.

Using the **left button** will result in the complete sprite to be filled up with the current color. And the **middle button** is the most powerful. If you click this, CA3 will create **pasteable Z80 data structures** for the **patterns** and **colors** of **this sprite**. This code can be pasted in your source and is **READY TO USE**!

3.7. Editing sources

3.7.1. MACRO'S.ASM ?

What is macro's.asm? You probably saw this filename a few times earlier in this help file.

The '**macro's.asm**' file is delivered with CA3 and has all kinds of standard defines and labels in it... With this **macro's.asm** file you can e.g. call **any bios call** (MSX1 or MSX2) and easily access the **BDOS** routines. It also has some macro's defined, stuff like **setting palettes**, **copying ram to vram**, **setting VRAM pages**, **reading sectors from disk** etc. etc.

You should really take a look at this file. This file also is BY DEFAULT included in a new source file. TeddyWareZ used it in all their products and it just is really comfortable. Also you can update this file just to let it fit your needs. This file really is the basis of a new file.

3.7.2. Code completion

You can of course just type away in the editor, and view all those nice colored keywords and numbers, strings and all other things, but you want more, now don't you? of course you want that... Get ready to be astonished.

What is code completion?
Code completion is never before shown in any MSX assembler. It does what you probably think it does, it completes code for you. Code completion comes in especially when you have long label names from which you can't remember the complete name anymore. Also for macro's, this can come in handy... The basis is that you type the first part of a label and call upon the **code completion**, the code completion will then make a list with ALL macro's and labels it can find (**EVEN the ones in an included file!**). After that list is made, code completion presents the list and will select the first item that corresponds to what you typed. After that (if you had the right label) you can press enter and code completion will fill in the label or macro on the appropriate place.
That's the basis of code completion. An example (see figure 44):

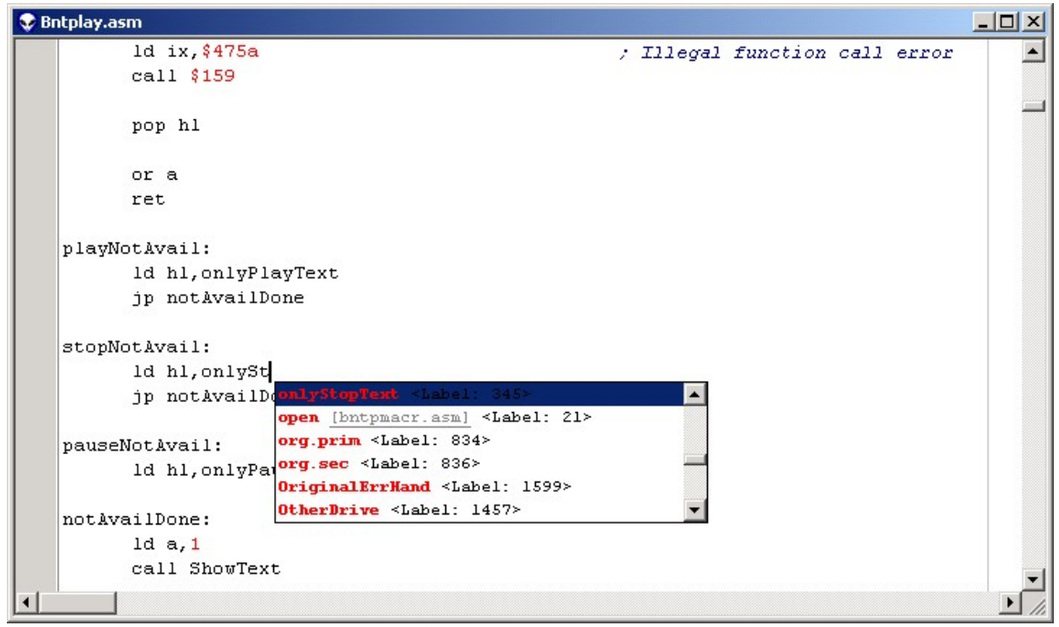

Figure 44. An example of code completion

Here you see the code completion in action. I typed **ld hl,onlySt**, cause I knew the label started with that. After that I pressed **ctrl+space** which invoked the code completion and the code completion did the rest. As you can see the code completion thinks it should be the label that's selected and in this case that's the right one. Pressing **enter** now will result in code completion completing my label. Actually it **replaces** my already typed part of the label, so that **case** sensitivity is also taken over. If i **kept typing** after the **onlySt** text, code completion would try to find a new **most appropriate** label and select it. Using the **arrow keys** will result in selecting another label from the list, just as **page up** and **page down**. Now that's what I call **FITTING THE PROGRAMMERS NEEDS**!
TeddyWareZ is VERY proud of this feature. Try looking at the figure. Look at the second item you see... You see that THAT label (**open**) is located in **bntpmacr.asm**! That's a completely **other file** just **included** in this source! Code completion will **scan that file too**!
You can also see at what line the label is found, so pressing **ALT+G** (Goto line) you can easily go to that line.

3.7.3. Code tool tips

Is **ld hl,(ix+1)** allowed? What does **neg** do? How fast is **scf**? How many bytes would **ld a,(ix+2)** take? These are just some questions you and I probably sometimes have. Though you maybe an experienced Z80 assembly programmer, you probably have Z80 instruction tables and timings on hard copy when you're coding. When you think of something that raises a question about the syntax or timing, you look it up in those papers.
This will be history with CA3. TeddyWareZ is proud to introduce **Code Tool Tips** (CTT's). With CTT's you can get all kinds of answers to your question... By pressing **ctrl+shift+space** after you typed (a part) of a MNEMONIC, you can give a call upon the CTT's to show you some info about that MNEMONIC as you can see in figure 45.

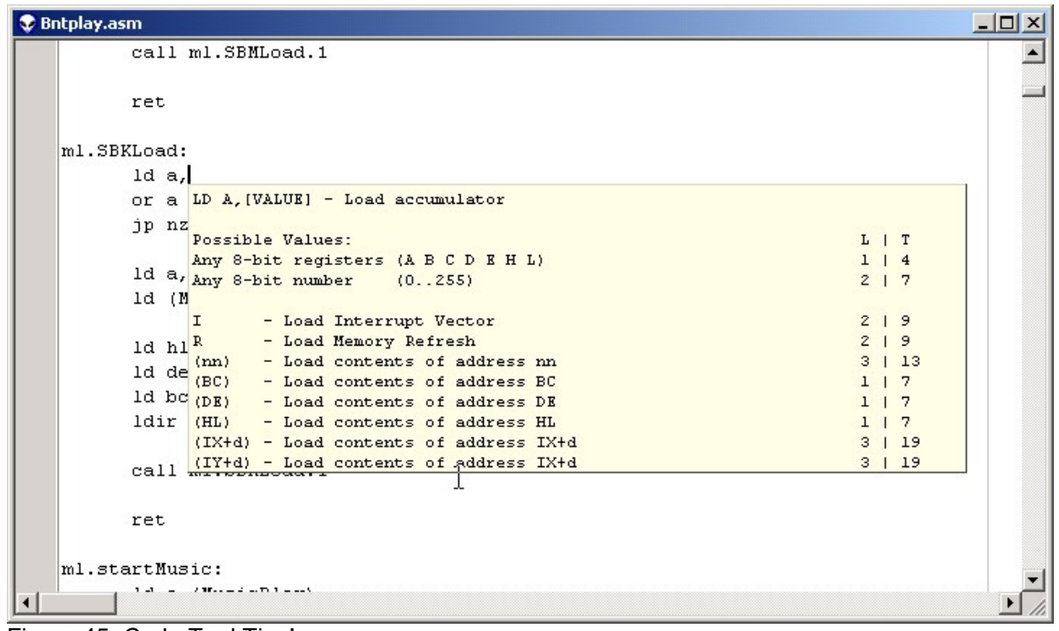

Figure 45. Code Tool Tips!

I don't think I have to tell really much. Notice the **L | T** structure... The **L** stands for Length and displays the **number of bytes** the MNEMONIC will take. The **T** stands for Time and displays the number of **T-states** that instruction will take.
That's it, have fun with it!

3.7.4. Other tool tips

Label recognition
CA3 has build in more tool tips like the CTT's discussed in the previous topic. One of these tool tips are the **label recognition** tips. CA3 can recognize labels and let you view what line these labels are... See figure 46.

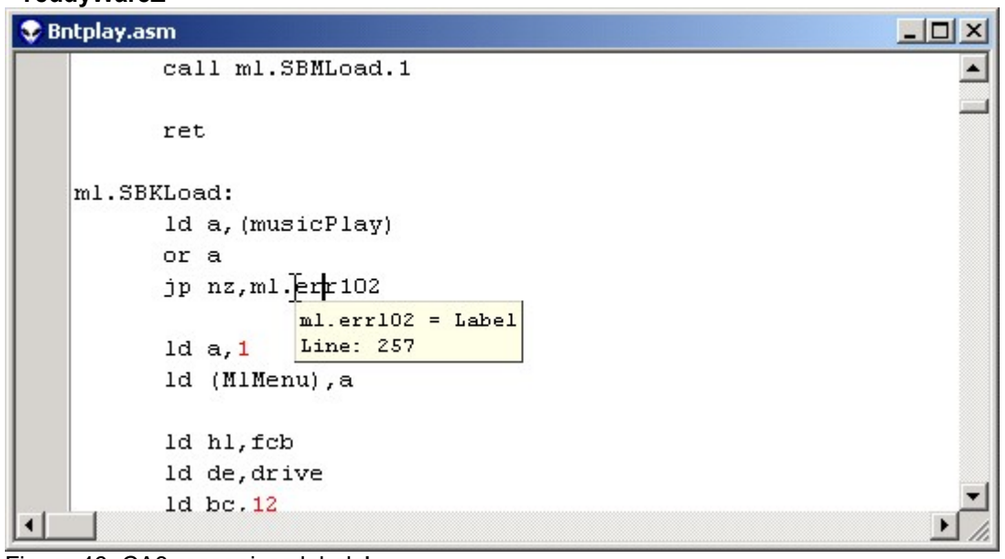

Figure 46. CA3 recognizes labels!

Here you can see the label recognition in action. As you can see the **ml.err102** label is located at line 257. The most powerful option of label recognition is yet to come. Try pressing **CTRL** while moving over the label. The **cursor** will change to a **hand point**. If you click now (whilst holding **CTRL**) CA3 will **jump** to the line where this label is declared! That is something REALLY powerful. Maybe even MORE powerful is this: After you've jumped to a label, try hitting **CTRL+BACKSPACE**. CA3 will then RETURN to the position you were BEFORE you jumped to the label!

File recognition

Another thing CA3 has built in is *file recognition*. This works almost the same as the label recognition, difference is this is for ***included*** files. Figure 47 explains better.

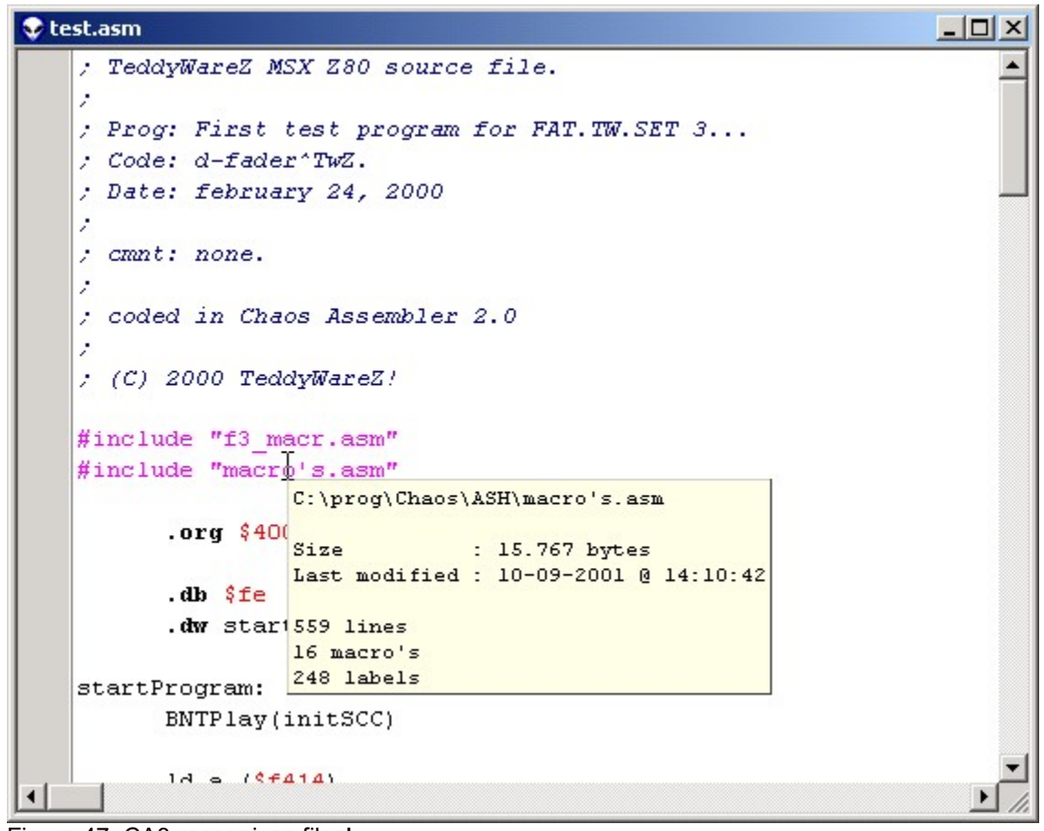

Figure 47. CA3 recognizes files!

Here you see the file recognition in action. It works exactly the same as the label recognition. **Move** your **mouse** over a **filename**. After a short while a **tool tip** will be shown with all kinds of **information** about the file! By holding **CTRL** and clicking when the mouse has changed to a **hand point** CA3 will open the file in a new window or pop-up the window if the file already is open. It's THAT simple...

3.7.5. Math evaluation

Yet another great feature of this editor is the math evaluation. It does exactly what you think it should do, it evaluates things. Mathematical things to be exact. The math evaluator will also pop-up just like the *other tool tips* and is actually a tool-tip too, but this one deserves a whole topic. Math evaluation is almost a must for every programmer around the world. How many times did you see a hexadecimal number and wanted to convert it to a decimal number or saw a binary number added to a hexadecimal number and you needed the result of that? Probably pretty often. Well CA3 will do this FOR you... Check figure 48 for that!

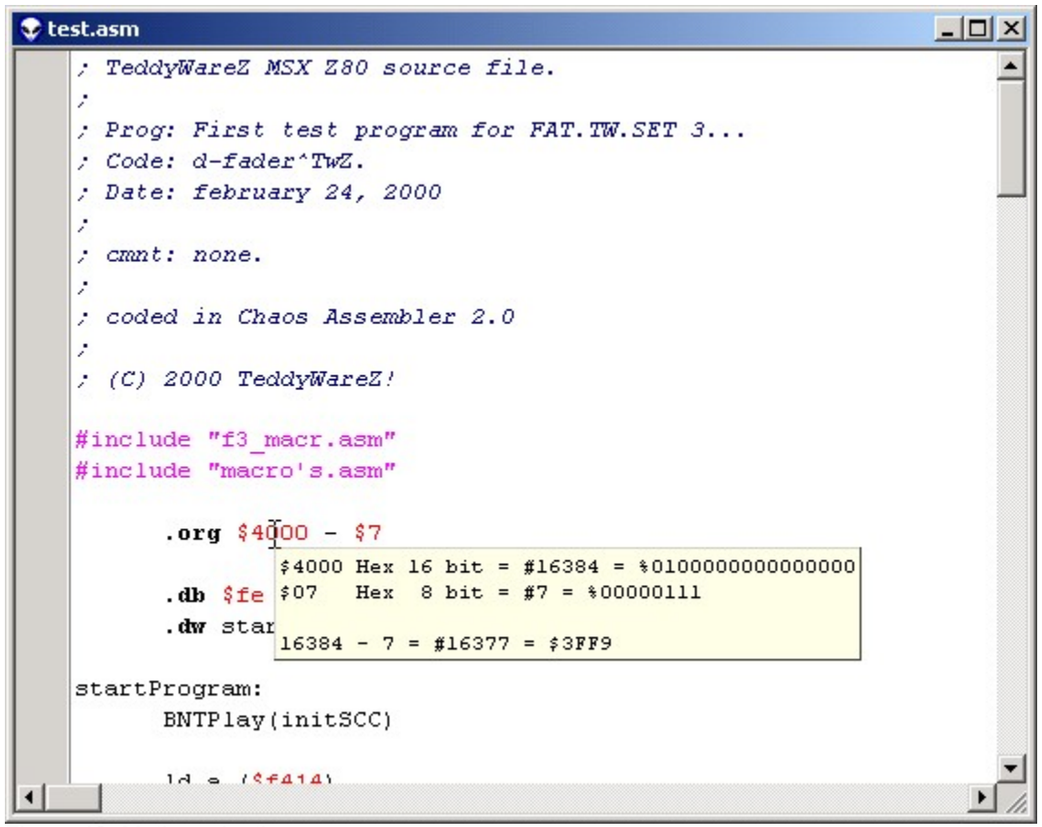

Figure 48. Math evaluation

here you see the math evaluation in action. As you can see CA3 will split up what it finds into small pieces of numbers first. CA3 will show all those numbers in **hexadecimal**, **decimal** and **binary**. After that the math evaluator comes in action. It'll calculate the sum of what you selected or what CA3 thinks it should calculate (In this case no text was selected and CA3 has determined that you want to see the result of **$4000 - $7**. At the bottom of the tool tip you see the sum and the result in decimal and hexadecimal...

As I told, if you select a part of text in the file, the math evaluator will only evaluate THAT part. This can come in handy if CA3 can't determine the sum or does it wrong (See figure 49)...

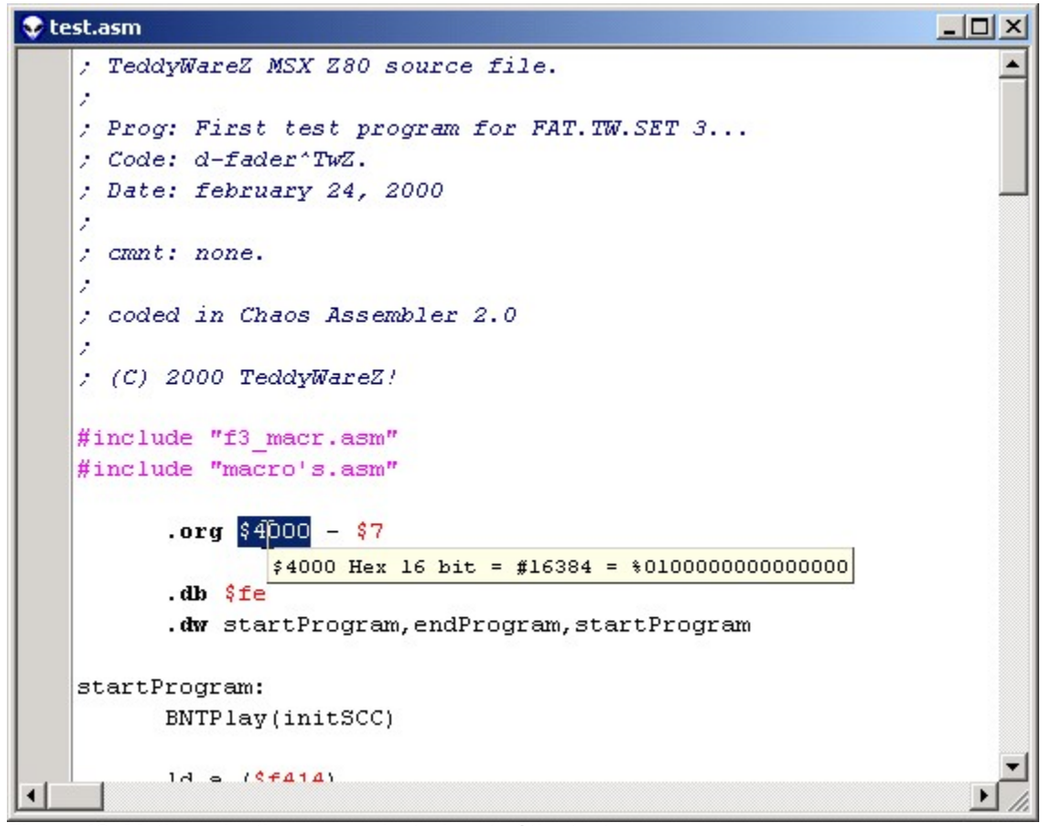

Figure 49. Math evaluation on a selected part of the text.

I didn't want the complete sum, I only wanted the **$4000**. By selecting that what you want and calling upon the evaluator, it'll evaluate the stuff you selected!! Quite powerful eh?

3.7.6. In general

Undo / Redo

The editor can undo your changes up to 1024 levels... You will probably use this REALLY often. The shortcut to undo a change is **CTRL+Z** and to redo the last undo you can press **CTRL+SHIFT+Z**. Undo and redo is also available on the edit childs **tool bar**, in the **main menu** and on the **pop-up menu** of the edit child.

The pop-up menu

In figure 50 you see the pop-up menu of the edit child. No explanation is necessary, cause it explains itself and every item in the menu is discussed extensively.

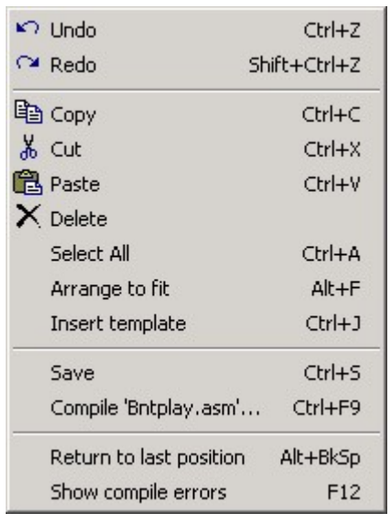

Figure 50. The edit child pop-up menu

The only thing I want to say is that you can show and hide the error list box shown when you're source had errors... By pressing **F12** or selecting the item in this pop-up you'll toggle the **compiler error list box**.

The edit child status bar

The edit child has an extensive status bar which gives you information whilst you are typing. It's divided into three parts, from which the second part is divided into three parts itself.

The first part (on the left side) gives you information about the cursor position. It shows the current cursor position in **horizontal**, **vertical** way. On the right side of this part you see a number between brackets... This number is the **ASCII** value of the character of the cursor. In figure 41 it's **49**. ASCII 49 = character '**1**' and as you can see in the image, the cursor is located at the **1** of **1999**...

The second group of status info's are the status of the three well-known keys **num-lock**, **caps-lock** and **scroll-lock**. If one of these lock keys is enabled, the text will be visible in the status bar. Else nothing will be shown.

The third part of the status bar has MORE than one purpose. If NO text is selected in the edit child, this part of the status bar will display the insert status. If it reads '**Insert**', then insert mode is on and no text will be overwritten. If this is '**Overwrite**' text will be overwritten.
If after this insert status it also reads '**(Modified)**' then the file you are editing is modified and thus save able (see figure 51).
On the other hand, if you have some text selected, this status bar will read HOW many bytes you have currently selected. As soon as no selection is available anymore, this will change to the default 'insert' status info.

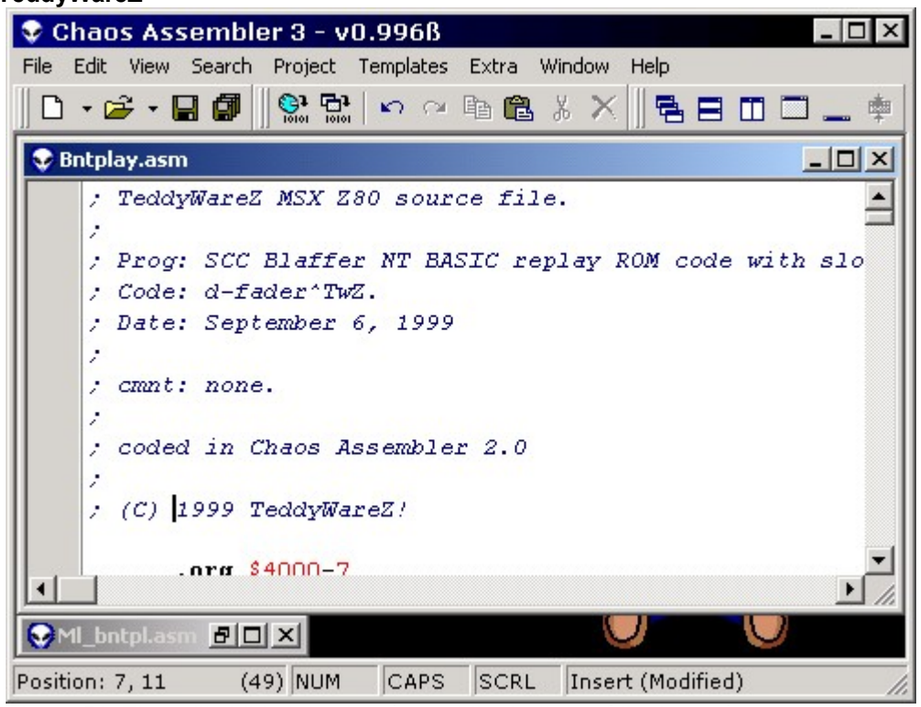

Figure 51. The edit child status bar (see at the bottom of the image)...

Bookmarks

CA3 support bookmarks. A bookmark is in fact a cursor position in your source. To make a bookmark go anywhere in your source, press **CTRL+SHIFT+[number]**. A visual confirmation you can see at the left side of the edit child. The visual confirmation (see figure 51) will be the number you pressed. You can only set one bookmark per line. Press **CTRL+SHIFT+[number]** again on the same line to REMOVE the bookmark. To jump to a specific bookmark, press **CTRL+[number]** where *[number]* again is the index of the bookmark. Try to get used to bookmarks, they are REALLY handy! Per edit child you can make 10 bookmarks (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9).

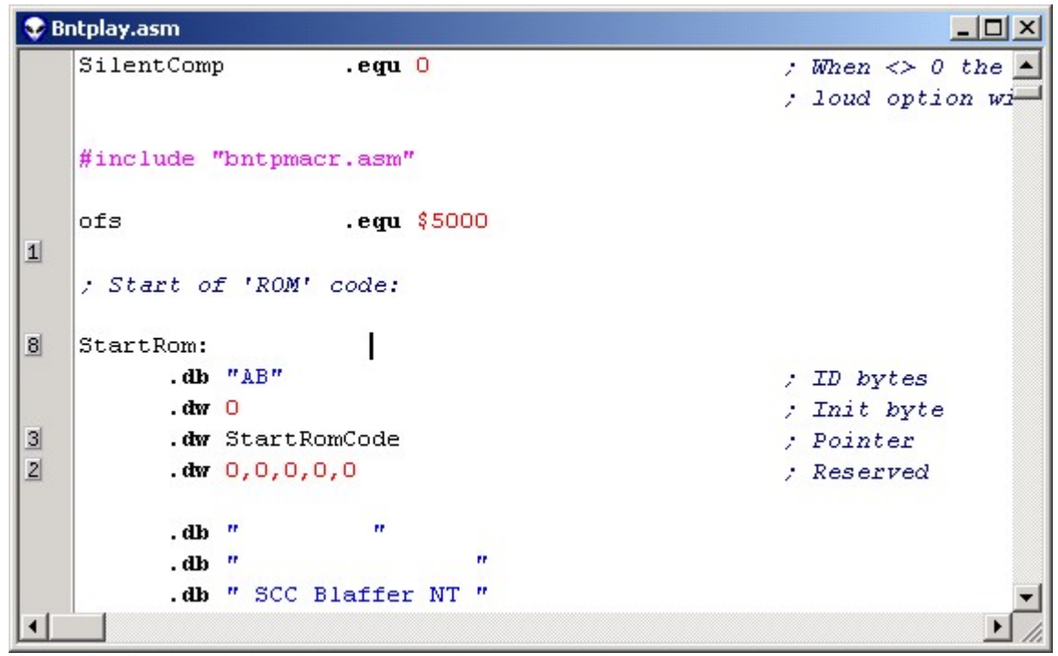

Figure 51. Bookmarks!

Code templates

Code templates are already pretty extensive described in the Code templates tab of the settings screen and the UNREGISTERED DEMO VERSION. This topic will describe how to implement the code templates in your source code. There's not much to it, but it just has to be described...

To insert a code template, type the shortcut of the code template and press **CTRL+J** to find the implementation of it and insert it in your code. Another way to implement a code template in your code is by selecting the one you need from the **Template** menu (see figure 52).

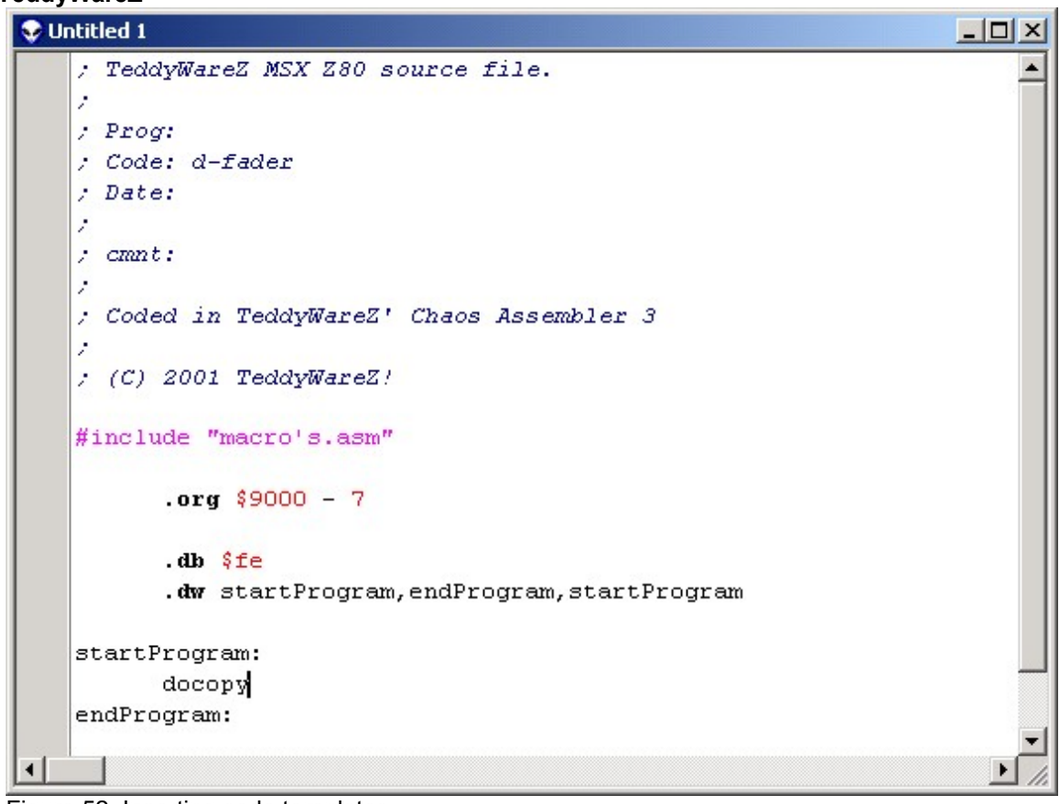

Figure 52. Inserting code templates.

By pressing **CTRL+J** at this moment in your source, the code template with **shortcut** *docopy* will be inserted into the source (if available that is). That's all I can tell about templates, they're just easy to use neat features!

4. Other

4.1. Trouble shooting

This is a pretty hard topic, as we don't know what problems users will have and this is the first version of the help file of CA3. Nevertheless we have found some things that fit in the trouble shooting section...

Problems using macro's

I'm sure I declared my macro correctly, but TASM gives an error like 'line 0023: unrecognized directive. (([Parameter name]))'
This is a problem we've encountered too... This is a bug (?) in TASM. To solve this be sure your definition of the macro does not contain any spaces between the name and the parameters... example:

```
#define MyMacro (Parameter)
```
This will generate an error, because there's a white space between the MyMacro and the '(' of the parameter. To define it correct, define it like this:

```
#define MyMacro(Parameter)
```
This is the correct definition of a macro.

I'm sure I declared my macro correctly, but TASM gives an error like 'line 0023: Macro expects args but none found'
This is almost the same problem as the problem described before, though in this case you DID define the macro correctly, but called upon the maco in a wrong way. You probably called your macro like this:

```
MyMacro (Parameter)
```
This will generate an error, because there's a white space between the MyMacro and the '(' of the parameter. To call it correct, implement it like this:

```
MyMacro(Parameter)
```
This is the correct implementation of calling upon a macro.

I'm sure I declared my macro correctly, but TASM gives an error like 'line 0023: unrecognized instruction. (MYMACRO(PARAMETER))'
This error occurs because the implemantation of macro's is CASE SENSITIVE, even if the option of case sensitivity is disabled! Check the case of the macro name you call upon...

Problems using labels

I'm sure I typed the labelname correctly, still I get an error like: 'line 0023: Label not found: (LabelName)'
This error occurs when you call upon a label with the wrong CASE SENSITIVITY if the '**Ignore case in labels**' option in the settings screen is DISABLED. To get rid of this error, go to the settings screen, click the assembler tab, enable the option 'Ignore case in labels' and click OK.

I'm getting errors like 'line 0023: label value misalligned. (jp)'
This error occurs when you do not have any WHITE SPACES at the left side of a line and directly typed an instruction like:

```
Label1:
jp Label1
```

This generates an error, because the **jp label1** is an instruction and didn't start with a white space. It is generally recommended to start lines with an instruction with a **TAB** character!

4.2. The TeddyWareZ recovery system

Another neat feature of CA3 is the TwZ recovery system. This one comes in action when CA3 becomes instable. Programs that become instable are a pretty issue since the introduction since Windows '95. Also, remember CA3 is really big and I can't help it, but I can assure you there are bugs in it like not freeing memory right etc. TeddyWareZ found a way to detect that and let you decide what to do. This includes that CA3 can save modified sources after it became instable (see figure 44).

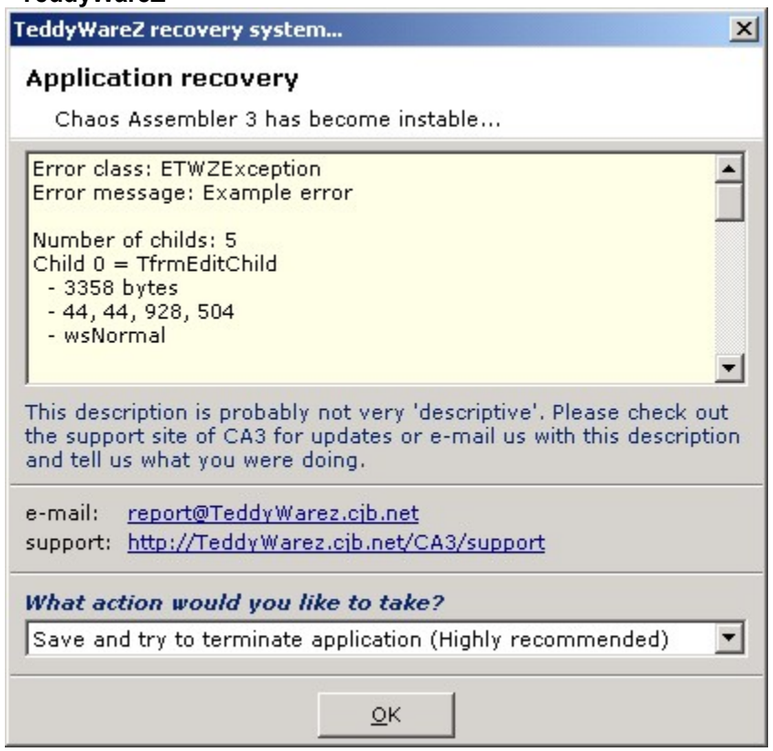

Figure 44. The TeddyWareZ recovery system.

If you get this screen, please check the support site for updates and / or e-mail us the description in the little text box. By just clicking on of the links you'll be warped to your e-mail client or your HTTP browser.

4.3. Tips & Hints

Hints

Almost everywhere in CA3 you can get **HINTS** and **TOOLTIPS**. These are to help you identify certain objects.. Just by moving the mouse over objects, you'll DIRECTLY see a hint (if available) in the status bar of the main window... Check out figure 53 to see what I mean.

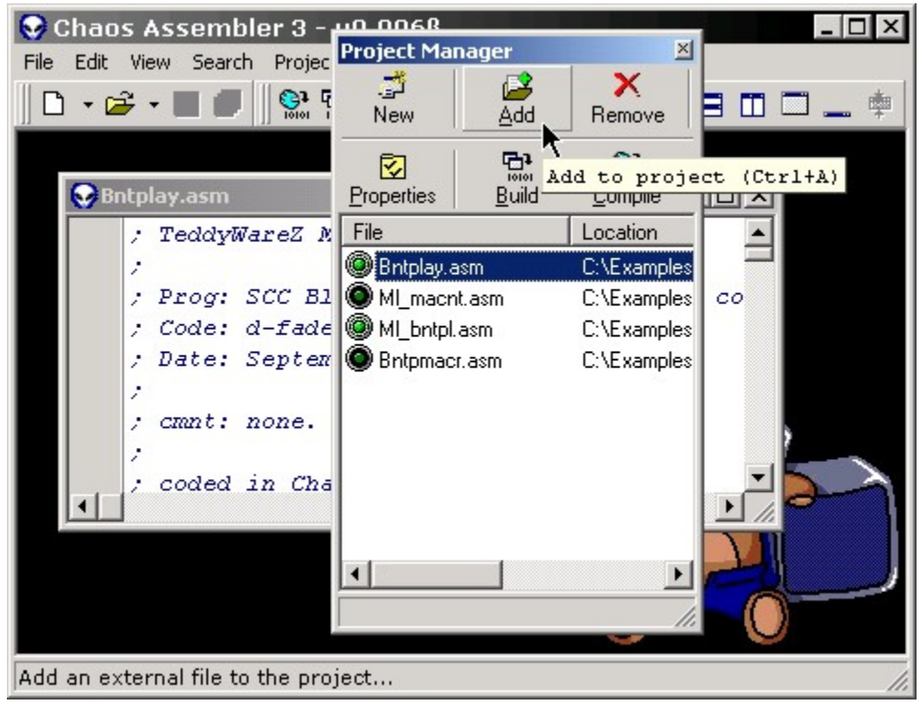

Figure 53. Hint and tool tip

Hot keys

I can not tell often enough that CA3 has a big amount of **hot keys** (also known as key combinations). Hot keys are there for you. Most items in the main menu of CA3 have a hot key. That means that those items can be called using the keyboard. e.g. The **Project > Compile 'filename'** menu item can be called by using the hotkey **CTRL+F9**... Get used to these key combinations. It is recommended to print out the key combinations section of this chapter...

DB, DW and DS

As TASM wants **db**, **dw** and **ds** as **.db**, **.dw** and **.fill**, adapting sources from other compilers is quite a lot of work. You can of course replace all occurrences of these directives to fit TASM, but then you'll lose the compatibility of MSX compilers. If you define these things then you can use DB, DW and DS in your sources and TASM will auto convert them to **.db**, **.dw** and **.fill** for you:

```
#DEFINE db .db
#DEFINE dw .dw
#DEFINE ds .fill
```

The only disadvantage is that the CA3 syntax highlighter will not recognize these defines as .db, .dw and .fill...

4.4. How to ...?

1) How to set the destination file?

To set the destination file for a file in the project, go to the project manager, select the file where you want to set the destination for and click the properties button. Now click on the little folder right to the text of the destination file.

2) What to do if an access voilation occurs?

If an access violation occurs, remember what you were doing and press OK. If CA3 after that becomes instable, the TeddyWareZ recovery system will take over.

3) How to dock the project manager?

To dock the project manager drag it to the border of the CA3 screen, and drop it.. You'll see the manager will dock to one of the borders.

4) How to get data from the sprite editor in your source?

There are 3 ways to get the sprite data onto the clipboard

- ***ctrl+c***
- ***The button in the sprite editor, (2 documents button)***
- ***Main menu item 'sprite editor', copy sprite to clipboard... The sprite editor has to be active to make this menu item become available.***

In your source you can press ctrl+v or, right-mouse+past, this will insert the data into your source.

5) How to get palette data in the image viewer?

There are 2 ways to get the palette data onto the clipboard

- ***right mouse click on the image + copy palette to clipboard***
- ***-shift+ctrl+c***

In your source you can press ctrl+v or, right-mouse+past, this will insert the data into your source.

4.5. Key combinations

Main window

CTRL+N	New document (new file)
CTRL+SHIFT+N	New document (new file) and add to project
CTRL+O	Open a file (Normal file or project)
CTRL+S	Save the active edit child
CTRL+P	Print the active edit child (will show a print preview first)
CTRL+F4	Close the active child (edit child or not)
CTRL+F12	Close all open files
CTRL+SHIFT+F12	Close all open files and close the project too
CTRL+ALT+S	Toggle standard tool bar
CTRL+ALT+E	Toggle edit tool bar
CTRL+ALT+W	Toggle window tool bar
F11	Toggle project manager
CTRL+F3	View last compiler output
F10	View all errors of the last compilation
CTRL+SHIFT+L	View latest list file of the active edit child

CTRL+SHIFT+ENTER Add external file to the project
CTRL+F1 Add active edit child to the project
CTRL+F2 Remove active edit child from the project
CTRL+F9 Compile the active edit child
F9 Build the active project
CTRL+SHIFT+E Edit code templates
CTRL+F8 Open a new sprite editor
CTRL+F10 Open a new image viewer
SHIFT+F11 Open the chaotic media player
ALT+S Open preferences / settings window
ALT+F Arrange to fit all open child windows
CTRL+DOWN Minimize active child window
CTRL+UP Maximize active child window
CTRL+= Restore active child window (i.e. normal size)

Edit child

CTRL+SPACE Call upon code completion
CTRL+SHIFT+SPACE Call upon code tool tip
CTRL+SHIFT+[number] Set bookmark [number] on current line ([number] = 1..0)
CTRL+ [NUMBER] Go to bookmark [number] ([number] = 1..0)
CTRL+J Call upon the code template implementation
CTRL+RIGHT Go to next word
CTRL+LEFT Go to previous word
CTRL+Z Undo last change (up to 1024 levels)
CTRL+SHIFT+Z Redo last undo
CTRL+X Cut selected text to clipboard
CTRL+C Copy selected text to clipboard
CTRL+V Paste selected text from clipboard
ALT+BACKSPACE Return to last position (before a label jump)
CTRL+F Find text in edit child
CTRL+SHIFT+F Find text in files (files in the active project and / or open files)
CTRL+R Find text in files and replace with other text
F3 Find next occurrence
SHIFT+F3 Find previous occurrence
CTRL+E Incremental search
ALT+G Go to line number...

The project manager

CTRL+UP Move selected file up (earlier build)
CTRL+DOWN Move selected file down (later build)
CTRL+SHIFT+UP Move selected file to top (first build)
CTRL+SHIFT+DOWN Move selected file to bottom (last build)
CTRL+N New file and add to project
CTRL+A Add external file to project
CTRL+DEL Remove selected file from the project
CTRL+ENTER Change file properties (selected file)
CTRL+F9 Compile selected file
F9 Build the project

The image viewer

ENTER Load image
CTRL+S Export (save) image on screen
CTRL+P Import palette
CTRL+I Zoom in on image
CTRL+U Zoom out on image
CTRL+D Zoom to default image size (zoom x1)
CTRL+1..8 Change palette to palette x (where x = 1 through 8).
CTRL+9 Call upon the palette editor
MOUSEWHEEL Scroll image vertically
CTRL+MOUSEWHEEL Scroll image horizontally
SHIFT+MOUSEWHEEL Zoom in or out on image
SHIFT+MOUSEMOVE Mark a block

The the sprite editor

ENTER Load image
CTRL+S Export (save) image on screen
CTRL+P Import palette
CTRL+I Zoom in on image
CTRL+U Zoom out on image
CTRL+D Zoom to default image size (zoom x1)
CTRL+1..8 Change palette to palette x (where x = 1 through 8).
ALT+I Display sprite information
CTRL+C Copy palette information to clipboard
ALT+C Copy sprite information to clipboard (pattern and color)

(SPRITE EDITING)
LEFT CLICK Change the current pixel color the index color
RIGHT CLICK Change the current pixel color to the index color

(COLOR SELECTION)
LEFT CLICK Change the index color to the color selected
RIGHT CLICK Change the intensity of the color selected

Chaotic Media Player

ENTER	Add songs to list
CTRL+LEFT	Go to first song in list
CTRL+RIGHT	Go to last song in list
Z	Play song
X	Stop playing
C	Pause / Continue playing
V	Go to previous song in list
B	Go to next song in list
DEL	Remove selected songs from the list

That's it...

4.6. Support

There's support in three ways. You can visit the CA3 section of our homepage, you could e-mail us and we're even available by snail mail!

WWW: http://TeddyWareZ.cjb.net/CA3
e-mail: info@TeddyWareZ.cjb.net

Snail mail:

TeddyWareZ
Jan Palachweg 17
9403 JS Assen
The Netherlands

5. Z80 Information

5.1. Instruction set

Registers

The following registers are available:

A, B, C, D, E	8-bit registers
AF	16-bit register containing A and flags
BC	16-bit register containing B and C (BC=Byte Counter)
DE	16-bit register containing D and E
HL	16-bit register use for addressing (HL=High/Low)
F	8-bit flag register
IX, IY	16-bit index registers
PC	16-bit Program Counter register
R	8-bit Memory Refresh register
SP	16-bit Stack Pointer register

Addressing Methods

The following addressing methods are available:

n	Immediate addressing (8-bit)
nn	Immediate extended addressing (16-bit)
e	Relative addressing (8-bit; PC=PC+2+offset)
[nn]	Extended addressing (16-bit)
[xx + d]	Indexed addressing (16-bit + 8-bit)
r	Register addressing (8-bit)
[rr]	Register indirect addressing (16-bit)
b	Bit addressing
p	Modified page 0 addressing

Flags

The following flags are available:

S	Sign flag (bit 7)
Z	Zero flag (bit 6)
H	Half carry flag (bit 4)
P	Parity/Overflow flag (bit 2)
N	Add/Subtract flag (bit 1)
C	Carry flag (bit 0)

Symbol Descriptions

These symbols are included in the instruction list.

b	One bit (0 to 7)
cc	Condition (C,M,NC,NZ,P,PE,PO,Z)
d	One-byte expression (-128 to +127)
dst	Destination s, ss, [BC}, [DE], [HL], [nn]
e	One-byte expression (-126 to +129)
m	Any register r, [HL] or [xx+d]
n	One-byte expression (0 to 255)
nn	Two-byte expression (0 to 65535)
pp	Register pair BC, DE, IX or SP
qq	Register pair AF, BC, DE or HL
qq'	Alternative register pair AF, BC, DE or HL
r	Register A, B, C, D, E, H or L
rr	Register pair BC, DE, IY or SP
s	Any register r, value n, [HL] or [xx+d]
src	Source s, ss, [BC], [DE], [HL], nn, [nn]
ss	Register pair BC, DE, HL or SP
xx	Index register IX or IY

Flag Field Values

The each flag field contains one of the following:

-	Flag unaffected
*	Flag affected
0	Flag reset
1	Flag set
?	Unknown

Instuction List

	Flags		
Mnemonic	**SZHPNC**	**Description**	**Notes**
`ADC A,s`	`***V0*`	`Add with carry`	`A=A+s+CY`

```
ADC HL,ss   **?V0*   Add with carry       HL=HL+ss+CY
ADD A,s  ***V0*   Add               A=A+s
ADD HL,ss   --?-0*   Add               HL=HL+ss
ADD IX,pp   --?-0*   Add               IX=IX+pp
ADD IY,rr   --?-0*   Add               IY=IY+rr
AND s    ***P00   Logical AND          A=A&s
BIT b,m  ?*1?0-   Test Bit          m&{2^b}
CALL cc,nn  ------   Conditional Call        If cc CALL
CALL nn  ------   Unconditional Call      -[SP]=PC, PC=nn
CCF      --?-0*   Complement Carry Flag      CY=-CY
CP s     ***V1*   Compare           A-s
CPD      ****1-   Compare and Decrement      A-[HL], HL=HL-1, BC=BC-1
CPDR     ****1-   Compare, Dec., Repeat      CPD 'til A=[HL] or BC=0
CPI      ****1-   Compare and Increment      A-[HL], HL=HL+1, BC=BC-1
CPIR     ****1-   Compare, Inc., Repeat      CPI 'til A=[HL] or BC=0
CPL      --1-1-   Complement           A=-A
DAA      ***P-*   Decimal Adjust Acc.     A=BCD format
DEC s    ***V1-   Decrement            s=s-1
DEC xx   ------   Decrement            xx=xx-1
DEC ss   ------   Decrement            ss=ss-1
DI       ------   Disable Interrupts
DJNZ e   ------   Dec., Jump Non-Zero     B=B-1 'til B=0
EI       ------   Enable Interrupts
EX (SP),HL  ------   Exchange          [SP]<->HL
EX (SP),xx  ------   Exchange          [SP]<->xx
EX AF,AF'   ------   Exchange          AF<->AF'
EX DE,HL ------   Exchange          DE<->HL
EXX      ------   Exchange          qq<->qq' (except AF)
HALT     ------   Halt
IM n     ------   Interrupt Mode       (n=0,1,2)
IN A,(n) ------   Input             A=[n]
IN r,(C) ***P0-   Input             r=[C]
INC r    ***V0-   Increment            r=r+1
INC (HL) ***V0-   Increment            [HL]=[HL]+1
INC xx   ------   Increment            xx=xx+1
INC (xx+d)  ***V0-   Increment            [xx+d]=[xx+d]+1
INC ss   ------   Increment            ss=ss+1
IND      ?*??1-   Input and Decrement     [HL]=[C], HL=HL-1, B=B-1
INDR     ?1??1-   Input, Dec., Repeat     IND 'til B=0
INI      ?*??1-   Input and Increment     [HL]=[C], HL=HL+1, B=B-1
INIR     ?1??1-   Input, Inc., Repeat     INI 'til B=0
JP (HL)  ------   Unconditional Jump      PC=[HL]
JP (xx)  ------   Unconditional Jump      PC=[xx]
JP nn    ------   Unconditional Jump      PC=nn
JP cc,nn ------   Conditional Jump        If cc JP
JR e     ------   Unconditional Jump      PC=PC+e
JR cc,e  ------   Conditional Jump        If cc JR (cc=C,NC,NZ,Z)
LD dst,src  ------   Load              dst=src
LD A,I   **0*0-   Load              A=i (i=I,R)
LDD      --0*0-   Load and Decrement      [DE]=[HL], HL=HL-1, B=B-1
LDDR     --000-   Load, Dec., Repeat      LDD 'til BC=0
LDI      --0*0-   Load and Increment      [DE]=[HL], HL=HL+1, B=B-1
LDIR     --000-   Load, Inc., Repeat      LDI 'til B=0
NEG      ***V1*   Negate            A=-A
NOP      ------   No Operation
OR s     ***P00   Logical Inclusive OR    A=A OR S
OTDR     ?1??1-   Output, Dec., Repeat    OUTD 'til B=0
OTIR     ?1??1-   Output, Inc., Repeat    OUTI 'til B=0
OUT (C),r   ------   Output            [C]=r
OUT (n),A   ------   Output            [n]=A
OUTD     ?*??1-   Output and Decrement    [C]=[HL], HL=HL-1, B=B-1
OUTI     ?*??1-   Output and Increment    [C]=[HL], HL=HL+1, B=B-1
POP xx   ------   Pop               xx=[SP]+
POP qq   ------   Pop               qq=[SP]+
PUSH xx  ------   Push              -[SP]=xx
PUSH qq  ------   Push              -[SP]=qq
RES b,m  ------   Reset bit            m=m&{-2^b}
RET      ------   Return            PC=[SP]+
RET cc   ------   Conditional Return      If cc RET
RETI     ------   Return from Interrupt      PC=[SP]+
RETN     ------   Return from NMI         PC=[SP]+
RL m     **0P0*   Rotate Left          m={CY,m}<-
RLA      --0-0*   Rotate Left Acc.        A={CY,m}<-
RLC m    **0P0*   Rotate Left Circular    m=m<-
RLCA     --0-0*   Rotate Left Circular    A=A<-
RLD      **0P0-   Rotate Left 4 bits      {A,[HL]}={A,[HL]}<-
RR m     **0P0*   Rotate Right         m=->{CY,m}
RRA      --0-0*   Rotate Right Acc.       A=->{CY,m}
RRC m    **0P0-   Rotate Right Circular      m=->m
RRCA     --0-0*   Rotate Right Circular      A=->A
RRD      **0P0-   Rotate Right 4 bits     {A,[HL]}=->{A,[HL]}
RST p    ------   Restart           (p=0h,8h,10h,...,38h)
SBC A,s  ***V1*   Subtract with Carry     A=A-s-CY
SBC HL,ss   **?V1*   Subtract with Carry     HL=HL-ss-CY
SCF      --0-01   Set Carry Flag       CY=1
SET b,m  ------   Set bit           m=m or {2^b}
SLA m    **0P0*   Shift Left Arithmetic      m=m*2
SRA m    **0P0*   Shift Right Arithmetic     m=m/2
SRL m    **0P0*   Shift Right Logical     m=->{0,m,CY}
SUB s    ***V1*   Subtract          A=A-s
```

XOR s ***P00 Logical Exclusive OR A=A xor s

5.2. Instruction timings

Mnemonic	Clock	Siz	OP-Code
ADC A,(HL)	7	1	8E
ADC A,(IX+N)	19	3	DD 8E XX
ADC A,(IY+N)	19	3	FD 8E XX
ADC A,r	4	1	88+rb
ADC A,N	7	2	CE XX
ADC HL,BC	15	2	ED 4A
ADC HL,DE	15	2	ED 5A
ADC HL,HL	15	2	ED 6A
ADC HL,SP	15	2	ED 7A
ADD A,(HL)	7	1	86
ADD A,(IX+N)	19	3	DD 86 XX
ADD A,(IY+N)	19	3	FD 86 XX
ADD A,r	4	1	80+rb
ADD A,N	7	2	C6 XX
ADD HL,BC	11	1	09

Mnemonic	Clock	Siz	OP-Code
ADD HL,DE	11	1	19
ADD HL,HL	11	1	29
ADD HL,SP	11	1	39
ADD IX,BC	15	2	DD 09
ADD IX,DE	15	2	DD 19
ADD IX,IX	15	2	DD 29
ADD IX,SP	15	2	DD 39
ADD IY,BC	15	2	FD 09
ADD IY,DE	15	2	FD 19
ADD IY,IY	15	2	FD 29
ADD IY,SP	15	2	FD 39
AND (HL)	7	1	A6
AND (IX+N)	19	3	DD A6 XX
AND (IY+N)	19	3	FD A6 XX
AND r	4	1	A0+rb
AND N	7	2	E6 XX

Mnemonic	Clock	Siz	OP-Code
BIT b,(HL)	12	2	CB 46+8*b
BIT b,(IX+N)	20	4	DD CB XX 46+8*b
BIT b,(IY+N)	20	4	FD CB XX 46+8*b
BIT b,r	8	2	CB 40+8*b+rb
CALL C,NN	17/1	3	DC XX XX
CALL M,NN	17/1	3	FC XX XX
CALL NC,NN	17/1	3	D4 XX XX
CALL NC,NN	17/1	3	D4 XX XX
CALL NN	17	3	CD XX XX
CALL NZ,NN	17/1	3	C4 XX XX
CALL P,NN	17/1	3	F4 XX XX
CALL PE,NN	17/1	3	EC XX XX
CALL PO,NN	17/1	3	E4 XX XX
CALL Z,NN	17/1	3	CC XX XX

Mnemonic	Clock	Siz	OP-Code
CCF	4	1	3F
CP (HL)	7	1	BE
CP (IX+N)	19	3	DD BE XX
CP (IY+N)	19	3	FD BE XX
CP r	4	1	B8+rb
CP N	7	2	FE XX
CPD	16	2	ED A9
CPDR	21/1	2	ED B9
CPI	16	2	ED A1
CPIR	21/1	2	ED B1
CPL	4	1	2F
DAA	4	1	27
DEC (HL)	11	1	35
DEC (IX+N)	23	3	DD 35 XX
DEC (IY+N)	23	3	FD 35 XX
DEC A	4	1	3D

Mnemonic	Clock	Siz	OP-Code
DEC B	4	1	05
DEC BC	6	1	0B

DEC C	4	1	0D
DEC D	4	1	15
DEC DE	6	1	1B
DEC E	4	1	1D
DEC H	4	1	25
DEC HL	6	1	2B
DEC IX	10	2	DD 2B
DEC IY	10	2	FD 2B
DEC L	4	2	2D
DEC SP	6	1	3B
DI	4	1	F3
DJNZ $+2	13/8	1	10
EI	4	1	FB
EX (SP),HL	19	1	E3
EX (SP),IX	23	2	DD E3

Mnemonic	Clock	Siz	OP-Code
EX (SP),IY	23	2	FD E3
EX AF,AF'	4	1	08
EX DE,HL	4	1	EB
EXX	4	1	D9
HALT	4	1	76
IM 0	8	2	ED 46
IM 1	8	2	ED 56
IM 2	8	2	ED 5E
IN A,(C)	12	2	ED 78
IN A,(N)	11	2	DB XX
IN B,(C)	12	2	ED 40
IN C,(C)	12	2	ED 48
IN D,(C)	12	2	ED 50
IN E,(C)	12	2	ED 58
IN H,(C)	12	2	ED 60
IN L,(C)	12	2	ED 68

Mnemonic	Clock	Siz	OP-Code
INC (HL)	11	1	34
INC (IX+N)	23	3	DD 34 XX
INC (IY+N)	23	3	FD 34 XX
INC A	4	1	3C
INC B	4	1	04
INC BC	6	1	03
INC C	4	1	0C
INC D	4	1	14
INC DE	6	1	13
INC E	4	1	1C
INC H	4	1	24
INC HL	6	1	23
INC IX	10	2	DD 23
INC IY	10	2	FD 23
INC L	4	1	2C
INC SP	6	1	33
IND	16	2	ED AA

Mnemonic	Clock	Siz	OP-Code
INDR	21/1	2	ED BA
INI	16	2	ED A2
INIR	21/1	2	ED B2
JP $NN	10	3	C3 XX XX
JP (HL)	4	1	E9
JP (IX)	8	2	DD E9
JP (IY)	8	2	FD E9
JP C,$NN	10/1	3	DA XX XX
JP M,$NN	10/1	3	FA XX XX
JP NC,$NN	10/1	3	D2 XX XX
JP NZ,$NN	10/1	3	C2 XX XX
JP P,$NN	10/1	3	F2 XX XX
JP PE,$NN	10/1	3	EA XX XX
JP PO,$NN	10/1	3	E2 XX XX
JP Z,$NN	10/1	3	CA XX XX

Mnemonic	Clock	Siz	OP-Code
JR $N+2	12	2	18 XX
JR C,$N+2	12/7	2	38 XX
JR NC,$N+2	12/7	2	30 XX
JR NZ,$N+2	12/7	2	20 XX
JR Z,$N+2	12/7	2	28 XX
LD (BC),A	7	1	02
LD (DE),A	7	1	12
LD (HL),r	7	1	70+rb
LD (HL),N	10	2	36 XX
LD (IX+N),r	19	3	DD 70+rb XX
LD (IX+N),N	19	4	DD 36 XX XX
LD (IY+N),r	19	3	FD 70+rb XX
LD (IY+N),N	19	4	FD 36 XX XX
LD (NN),A	13	3	32 XX XX
LD (NN),BC	20	4	ED 43 XX XX

Mnemonic	Clock	Siz	OP-Code
LD (NN),DE	20	4	ED 53 XX XX
LD (NN),HL	16	3	22 XX XX

LD (NN),IX	20	4	DD 22 XX XX
LD (NN),IY	20	4	FD 22 XX XX
LD (NN),SP	20	4	ED 73 XX XX
LD A,(BC)	7	1	0A
LD A,(DE)	7	1	1A
LD A,(HL)	7	1	7E
LD A,(IX+N)	19	3	DD 7E XX
LD A,(IY+N)	19	3	FD 7E XX
LD A,(NN)	13	3	3A XX XX
LD A,r	4	1	78+rb
LD A,I	9	2	ED 57
LD A,N	7	2	3E XX
LD B,(HL)	7	1	46

Mnemonic	Clock	Siz	OP-Code
LD B,(IX+N)	19	3	DD 46 XX
LD B,(IY+N)	19	3	FD 46 XX
LD B,r	4	1	40+rb
LD B,N	7	2	06 XX
LD BC,(NN)	20	4	ED 4B XX XX
LD BC,NN	10	3	01 XX XX
LD C,(HL)	7	1	4E
LD C,(IX+N)	19	3	DD 4E XX
LD C,(IY+N)	19	3	FD 4E XX
LD C,r	4	1	48+rb
LD C,N	7	2	0E XX
LD D,(HL)	7	1	56
LD D,(IX+N)	19	3	DD 56 XX
LD D,(IY+N)	19	3	FD 56 XX
LD D,r	4	1	50+rb

Mnemonic	Clock	Siz	OP-Code
LD D,N	7	2	16 XX
LD DE,(NN)	20	4	ED 5B XX XX
LD DE,NN	10	3	11 XX XX
LD E,(HL)	7	1	5E
LD E,(IX+N)	19	3	DD 5E XX
LD E,(IY+N)	19	3	FD 5E XX
LD E,r	4	1	58+rb
LD E,N	7	2	1E XX
LD H,(HL)	7	1	66
LD H,(IX+N)	19	3	DD 66 XX
LD H,(IY+N)	19	3	FD 66 XX
LD H,r	4	1	60+rb
LD H,N	7	2	26 XX
LD HL,(NN)	20	3	2A XX XX
LD HL,NN	10	3	21 XX XX

Mnemonic	Clock	Siz	OP-Code
LD I,A	9	2	ED 47
LD IX,(NN)	20	4	DD 2A XX XX
LD IX,NN	14	4	DD 21 XX XX
LD IY,(NN)	20	4	FD 2A XX XX
LD IY,NN	14	4	FD 21 XX XX
LD L,(HL)	7	1	6E
LD L,(IX+N)	19	3	DD 6E XX
LD L,(IY+N)	19	3	FD 6E XX
LD L,r	4	1	68+rb
LD L,N	7	2	2E XX
LD SP,(NN)	20	4	ED 7B XX XX
LD SP,HL	6	1	F9
LD SP,IX	10	2	DD F9
LD SP,IY	10	2	FD F9
LD SP,NN	10	3	31 XX XX

Mnemonic	Clock	Siz	OP-Code
LDD	16	2	ED A8
LDDR	21/1	2	ED B8
LDI	16	2	ED A0
LDIR	21/1	2	ED B0
NEG	8	2	ED 44
NOP	4	1	00
OR (HL)	7	1	B6
OR (IX+N)	19	3	DD B6 XX
OR (IY+N)	19	3	FD B6 XX
OR r	4	1	B0+rb
OR N	7	2	F6 XX
OTDR	21/1	2	ED BB
OTIR	21/1	2	ED B3
OUT (C),A	12	2	ED 79
OUT (C),B	12	2	ED 41
OUT (C),C	12	2	ED 49

Mnemonic	Clock	Siz	OP-Code
OUT (C),D	12	2	ED 51
OUT (C),E	12	2	ED 59
OUT (C),H	12	2	ED 61
OUT (C),L	12	2	ED 69
OUT (N),A	11	2	D3 XX
OUTD	16	2	ED AB

OUTI	16	2	ED A3
POP AF	10	1	F1
POP BC	10	1	C1
POP DE	10	1	D1
POP HL	10	1	E1
POP IX	14	2	DD E1
POP IY	14	2	FD E1
PUSH AF	11	1	F5
PUSH BC	11	1	C5
PUSH DE	11	1	D5
PUSH HL	11	1	E5

Mnemonic	Clock	Siz	OP-Code
PUSH IX	15	2	DD E5
PUSH IY	15	2	FD E5
RES b,(HL)	15	2	CB 86+8*b
RES b,(IX+N)	23	4	DD CB XX 86+8*b
RES b,(IY+N)	23	4	FD CB XX 86+8*b
RES b,r	8	2	CB 80+8*b+rb
RET	10	1	C9
RET C	11/5	1	D8
RET M	11/5	1	F8
RET NC	11/5	1	D0
RET NZ	11/5	1	C0
RET P	11/5	1	F0
RET PE	11/5	1	E8
RET PO	11/5	1	E0
RET Z	11/5	1	C8
RETI	14	2	ED 4D

Mnemonic	Clock	Siz	OP-Code
RETN	14	2	ED 45
RL (HL)	15	2	CB 16
RL r	8	2	CB 10+rb
RL (IX+N)	23	4	DD CB XX 16
RL (IY+N)	23	4	FD CB XX 16
RLA	4	1	17
RLC (HL)	15	2	CB 06
RLC (IX+N)	23	4	DD CB XX 06
RLC (IY+N)	23	4	FD CB XX 06
RLC r	8	2	CB 00+rb
RLCA	4	1	07
RLD	18	2	ED 6F
RR (HL)	15	2	CB 1E
RR r	8	2	CB 18+rb
RR (IX+N)	23	4	DD CB XX 1E

Mnemonic	Clock	Siz	OP-Code
RR (IY+N)	23	4	FD CB XX 1E
RRA	4	1	1F
RRC (HL)	15	2	CB 0E
RRC (IX+N)	23	4	DD CB XX 0E
RRC (IY+N)	23	4	FD CB XX 0E
RRC r	8	2	CB 08+rb
RRCA	4	1	0F
RRD	18	2	ED 67
RST 0	11	1	C7
RST 8H	11	1	CF
RST 10H	11	1	D7
RST 18H	11	1	DF
RST 20H	11	1	E7
RST 28H	11	1	EF
RST 30H	11	1	F7
RST 38H	11	1	FF

Mnemonic	Clock	Siz	OP-Code
SBC (HL)	7	1	9E
SBC A,(IX+N)	19	3	DD 9E XX
SBC A,(IY+N)	19	3	FD 9E XX
SBC A,N	7	2	DE XX
SBC r	4	1	98+rb
SBC HL,BC	15	2	ED 42
SBC HL,DE	15	2	ED 52
SBC HL,HL	15	2	ED 62
SBC HL,SP	15	2	ED 72
SCF	4	1	37
SET b,(HL)	15	2	CB C6+8*b
SET b,(IX+N)	23	4	DD CB XX C6+8*b
SET b,(IY+N)	23	4	FD CB XX C6+8*b
SET b,r	8	2	CB C0+8*b+rb
SLA (HL)	15	2	CB 26

Mnemonic	Clock	Siz	OP-Code
SLA (IX+N)	23	4	DD CB XX 26
SLA (IY+N)	23	4	FD CB XX 26
SLA r	8	2	CB 20+rb
SRA (HL)	15	2	CB 2E
SRA (IX+N)	23	4	DD CB XX 2E
SRA (IY+N)	23	4	FD CB XX 2E
SRA r	8	2	CB 28+rb

```
SRL (HL)        15     2    CB 3E
SRL (IX+N)      23     4    DD CB XX 3E
SRL (IY+N)      23     4    FD CB XX 3E
SRL r           8      2    CB 38+rb
SUB (HL)        7      1    96
SUB (IX+N)      19     3    DD 96 XX
SUB (IY+N)      19     3    FD 96 XX
```

Mnemonic	Clock	Siz	OP-Code
SUB r	4	1	90+rb
SUB N	7	2	D6 XX
XOR (HL)	7	1	AE
XOR (IX+N)	19	3	DD AE XX
XOR (IY+N)	19	3	FD AE XX
XOR r	4	1	A8+rb
XOR N	7	2	EE XX

r means register (can be A, B, C, D, E, H, L).

Add this to last byte of OP-code:

Reg	Regbits
A	7
B	0
C	1
D	2
E	3
H	4
L	5

On >LD (IX+N),r< and >LD (IY+N),r< you add it to the byte before the last.

b means bit. Can be 0-7. Increase the last byte of OP-code with 8*b. Used in SET, BIT and RES.

If there is two numbers given at Clock, then the highest is when the jump is taken.

Collected by Oscar Lindberg 960324
(UNREGISTERED DEMO VERSION) from:

Z80 pocketbook
Z80 assembly language programming

5.3. Complete opcode list

This should be the complete list of all the opcodes of the Zilog Z80.

If an EDxx instruction is not listed, it should operate as two NOPs. If a DDxx or FDxx instruction is not listed, it should operate as without the DD or FD prefix, and the DD or FD prefix should operate as a NOP.

*** means unofficial.**

LD A,RLC (IX+d) means that the result of RLC (IX+d) is not only stored in (IX+d), but also in A. Double value for money. :)

SLL x operates the same as SLA x, except that SLL inserts 1 to the left.

OUT (C),0 always outs zero.

IN (C) / IN F,(C) does not store the result from the input. It only affects the flags, as the other IN r,(C) instructions do. These two notations both refer to the same unofficial instruction.

IM 0/1 sets the Z80 in IM 0 or in IM 1 mode.

RETN/I is either a RETI or a RETN.

```
OPCODE   MNEMONIC
00           NOP
01 n n       LD BC,nn
02           LD (BC),A
03           INC BC
04           INC B
05           DEC B
06 n         LD B,n
07           RLCA
08           EX AF,AF-
```

```
09          ADD HL,BC
0A          LD A,(BC)
0B          DEC BC
0C          INC C
0D          DEC C
0E n        LD C,n
0F          RRCA
10 d        DJNZ PC+d
11 n n      LD DE,nn
12          LD (DE),A
13          INC DE
14          INC D
15          DEC D
16 n        LD D,n
17          RLA
18 d        JR PC+d
19          ADD HL,DE
1A          LD A,(DE)
1B          DEC DE
1C          INC E
1D          DEC E
1E n        LD E,n
1F          RRA
20 d        JR NZ,PC+d
21 n n      LD HL,nn
22 n n      LD (nn),HL
23          INC HL
24          INC H
25          DEC H
26 n        LD H,n
27          DAA
28 d        JR Z,PC+d
29          ADD HL,HL
2A n n      LD HL,(nn)
2B          DEC HL
2C          INC L
2D          DEC L
2E n        LD L,n
2F          CPL
30 d        JR NC,PC+d
31 n n      LD SP,nn
32 n n      LD (nn),A
33          INC SP
34          INC (HL)
35          DEC (HL)
36 n        LD (HL),n
37          SCF
38 d        JR C,PC+d
39          ADD HL,SP
3A n n      LD A,(nn)
3B          DEC SP
3C          INC A
3D          DEC A
3E n        LD A,n
3F          CCF
40          LD B,B
41          LD B,C
42          LD B,D
43          LD B,E
44          LD B,H
45          LD B,L
46          LD B,(HL)
47          LD B,A
48          LD C,B
49          LD C,C
4A          LD C,D
4B          LD C,E
4C          LD C,H
4D          LD C,L
4E          LD C,(HL)
4F          LD C,A
50          LD D,B
51          LD D,C
52          LD D,D
53          LD D,E
54          LD D,H
55          LD D,L
56          LD D,(HL)
57          LD D,A
58          LD E,B
59          LD E,C
5A          LD E,D
5B          LD E,E
5C          LD E,H
5D          LD E,L
5E          LD E,(HL)
5F          LD E,A
60          LD H,B
61          LD H,C
62          LD H,D
```

```
63          LD H,E
64          LD H,H
65          LD H,L
66          LD H,(HL)
67          LD H,A
68          LD L,B
69          LD L,C
6A          LD L,D
6B          LD L,E
6C          LD L,H
6D          LD L,L
6E          LD L,(HL)
6F          LD L,A
70          LD (HL),B
71          LD (HL),C
72          LD (HL),D
73          LD (HL),E
74          LD (HL),H
75          LD (HL),L
76          HALT
77          LD (HL),A
78          LD A,B
79          LD A,C
7A          LD A,D
7B          LD A,E
7C          LD A,H
7D          LD A,L
7E          LD A,(HL)
7F          LD A,A
80          ADD A,B
81          ADD A,C
82          ADD A,D
83          ADD A,E
84          ADD A,H
85          ADD A,L
86          ADD A,(HL)
87          ADD A,A
88          ADC A,B
89          ADC A,C
8A          ADC A,D
8B          ADC A,E
8C          ADC A,H
8D          ADC A,L
8E          ADC A,(HL)
8F          ADC A,A
90          SUB B
91          SUB C
92          SUB D
93          SUB E
94          SUB H
95          SUB L
96          SUB (HL)
97          SUB A
98          SBC A,B
99          SBC A,C
9A          SBC A,D
9B          SBC A,E
9C          SBC A,H
9D          SBC A,L
9E          SBC A,(HL)
9F          SBC A,A
A0          AND B
A1          AND C
A2          AND D
A3          AND E
A4          AND H
A5          AND L
A6          AND (HL)
A7          AND A
A8          XOR B
A9          XOR C
AA          XOR D
AB          XOR E
AC          XOR H
AD          XOR L
AE          XOR (HL)
AF          XOR A
B0          OR B
B1          OR C
B2          OR D
B3          OR E
B4          OR H
B5          OR L
B6          OR (HL)
B7          OR A
B8          CP B
B9          CP C
BA          CP D
BB          CP E
BC          CP H
```

```
BD          CP L
BE          CP (HL)
BF          CP A
C0          RET NZ
C1          POP BC
C2 n n      JP NZ,nn
C3 n n      JP nn
C4 n n      CALL NZ,nn
C5          PUSH BC
C6 n        ADD A,n
C7          RST 0h
C8          RET Z
C9          RET
CA n n      JP Z,nn
CB00        RLC B
CB01        RLC C
CB02        RLC D
CB03        RLC E
CB04        RLC H
CB05        RLC L
CB06        RLC (HL)
CB07        RLC A
CB08        RRC B
CB09        RRC C
CB0A        RRC D
CB0B        RRC E
CB0C        RRC H
CB0D        RRC L
CB0E        RRC (HL)
CB0F        RRC A
CB10        RL B
CB11        RL C
CB12        RL D
CB13        RL E
CB14        RL H
CB15        RL L
CB16        RL (HL)
CB17        RL A
CB18        RR B
CB19        RR C
CB1A        RR D
CB1B        RR E
CB1C        RR H
CB1D        RR L
CB1E        RR (HL)
CB1F        RR A
CB20        SLA B
CB21        SLA C
CB22        SLA D
CB23        SLA E
CB24        SLA H
CB25        SLA L
CB26        SLA (HL)
CB27        SLA A
CB28        SRA B
CB29        SRA C
CB2A        SRA D
CB2B        SRA E
CB2C        SRA H
CB2D        SRA L
CB2E        SRA (HL)
CB2F        SRA A
CB30        SLL B*
CB31        SLL C*
CB32        SLL D*
CB33        SLL E*
CB34        SLL H*
CB35        SLL L*
CB36        SLL (HL)*
CB37        SLL A*
CB38        SRL B
CB39        SRL C
CB3A        SRL D
CB3B        SRL E
CB3C        SRL H
CB3D        SRL L
CB3E        SRL (HL)
CB3F        SRL A
CB40        BIT 0,B
CB41        BIT 0,C
CB42        BIT 0,D
CB43        BIT 0,E
CB44        BIT 0,H
CB45        BIT 0,L
CB46        BIT 0,(HL)
CB47        BIT 0,A
CB48        BIT 1,B
CB49        BIT 1,C
CB4A        BIT 1,D
CB4B        BIT 1,E
```

```
CB4C        BIT 1,H
CB4D        BIT 1,L
CB4E        BIT 1,(HL)
CB4F        BIT 1,A
CB50        BIT 2,B
CB51        BIT 2,C
CB52        BIT 2,D
CB53        BIT 2,E
CB54        BIT 2,H
CB55        BIT 2,L
CB56        BIT 2,(HL)
CB57        BIT 2,A
CB58        BIT 3,B
CB59        BIT 3,C
CB5A        BIT 3,D
CB5B        BIT 3,E
CB5C        BIT 3,H
CB5D        BIT 3,L
CB5E        BIT 3,(HL)
CB5F        BIT 3,A
CB60        BIT 4,B
CB61        BIT 4,C
CB62        BIT 4,D
CB63        BIT 4,E
CB64        BIT 4,H
CB65        BIT 4,L
CB66        BIT 4,(HL)
CB67        BIT 4,A
CB68        BIT 5,B
CB69        BIT 5,C
CB6A        BIT 5,D
CB6B        BIT 5,E
CB6C        BIT 5,H
CB6D        BIT 5,L
CB6E        BIT 5,(HL)
CB6F        BIT 5,A
CB70        BIT 6,B
CB71        BIT 6,C
CB72        BIT 6,D
CB73        BIT 6,E
CB74        BIT 6,H
CB75        BIT 6,L
CB76        BIT 6,(HL)
CB77        BIT 6,A
CB78        BIT 7,B
CB79        BIT 7,C
CB7A        BIT 7,D
CB7B        BIT 7,E
CB7C        BIT 7,H
CB7D        BIT 7,L
CB7E        BIT 7,(HL)
CB7F        BIT 7,A
CB80        RES 0,B
CB81        RES 0,C
CB82        RES 0,D
CB83        RES 0,E
CB84        RES 0,H
CB85        RES 0,L
CB86        RES 0,(HL)
CB87        RES 0,A
CB88        RES 1,B
CB89        RES 1,C
CB8A        RES 1,D
CB8B        RES 1,E
CB8C        RES 1,H
CB8D        RES 1,L
CB8E        RES 1,(HL)
CB8F        RES 1,A
CB90        RES 2,B
CB91        RES 2,C
CB92        RES 2,D
CB93        RES 2,E
CB94        RES 2,H
CB95        RES 2,L
CB96        RES 2,(HL)
CB97        RES 2,A
CB98        RES 3,B
CB99        RES 3,C
CB9A        RES 3,D
CB9B        RES 3,E
CB9C        RES 3,H
CB9D        RES 3,L
CB9E        RES 3,(HL)
CB9F        RES 3,A
CBA0        RES 4,B
CBA1        RES 4,C
CBA2        RES 4,D
CBA3        RES 4,E
CBA4        RES 4,H
CBA5        RES 4,L
```

```
CBA6        RES 4,(HL)
CBA7        RES 4,A
CBA8        RES 5,B
CBA9        RES 5,C
CBAA        RES 5,D
CBAB        RES 5,E
CBAC        RES 5,H
CBAD        RES 5,L
CBAE        RES 5,(HL)
CBAF        RES 5,A
CBB0        RES 6,B
CBB1        RES 6,C
CBB2        RES 6,D
CBB3        RES 6,E
CBB4        RES 6,H
CBB5        RES 6,L
CBB6        RES 6,(HL)
CBB7        RES 6,A
CBB8        RES 7,B
CBB9        RES 7,C
CBBA        RES 7,D
CBBB        RES 7,E
CBBC        RES 7,H
CBBD        RES 7,L
CBBE        RES 7,(HL)
CBBF        RES 7,A
CBC0        SET 0,B
CBC1        SET 0,C
CBC2        SET 0,D
CBC3        SET 0,E
CBC4        SET 0,H
CBC5        SET 0,L
CBC6        SET 0,(HL)
CBC7        SET 0,A
CBC8        SET 1,B
CBC9        SET 1,C
CBCA        SET 1,D
CBCB        SET 1,E
CBCC        SET 1,H
CBCD        SET 1,L
CBCE        SET 1,(HL)
CBCF        SET 1,A
CBD0        SET 2,B
CBD1        SET 2,C
CBD2        SET 2,D
CBD3        SET 2,E
CBD4        SET 2,H
CBD5        SET 2,L
CBD6        SET 2,(HL)
CBD7        SET 2,A
CBD8        SET 3,B
CBD9        SET 3,C
CBDA        SET 3,D
CBDB        SET 3,E
CBDC        SET 3,H
CBDD        SET 3,L
CBDE        SET 3,(HL)
CBDF        SET 3,A
CBE0        SET 4,B
CBE1        SET 4,C
CBE2        SET 4,D
CBE3        SET 4,E
CBE4        SET 4,H
CBE5        SET 4,L
CBE6        SET 4,(HL)
CBE7        SET 4,A
CBE8        SET 5,B
CBE9        SET 5,C
CBEA        SET 5,D
CBEB        SET 5,E
CBEC        SET 5,H
CBED        SET 5,L
CBEE        SET 5,(HL)
CBEF        SET 5,A
CBF0        SET 6,B
CBF1        SET 6,C
CBF2        SET 6,D
CBF3        SET 6,E
CBF4        SET 6,H
CBF5        SET 6,L
CBF6        SET 6,(HL)
CBF7        SET 6,A
CBF8        SET 7,B
CBF9        SET 7,C
CBFA        SET 7,D
CBFB        SET 7,E
CBFC        SET 7,H
CBFD        SET 7,L
CBFE        SET 7,(HL)
CBFF        SET 7,A
```

```
CC n n      CALL Z,nn
CD n n      CALL nn
CE n        ADC A,n
CF          RST 8h
D0          RET NC
D1          POP DE
D2 n n      JP NC,nn
D3 n        OUT (n),A
D4 n n      CALL NC,nn
D5          PUSH DE
D6 n        SUB n
D7          RST 10h
D8          RETC
D9          EXX
DA n n      JP C,nn
DB n        IN A,(n)
DC n n      CALL C,nn
DD09        ADD IX,BC
DD19        ADD IX,DE
DD21 n n    LD IX,nn
DD22 n n    LD (nn),IX
DD23        INC IX
DD24        INC IXH*
DD25        DEC IXH*
DD26 n      LD IXH,n*
DD29        ADD IX,IX
DD2A n n    LD IX,(nn)
DD2B        DEC IX
DD2C        INC IXL*
DD2D        DEC IXL*
DD2E n      LD IXL,n*
DD34 d      INC (IX+d)
DD35 d      DEC (IX+d)
DD36 d n    LD (IX+d),n
DD39        ADD IX,SP
DD44        LD B,IXH*
DD45        LD B,IXL*
DD46 d      LD B,(IX+d)
DD4C        LD C,IXH*
DD4D        LD C,IXL*
DD4E d      LD C,(IX+d)
DD54        LD D,IXH*
DD55        LD D,IXL*
DD56 d      LD D,(IX+d)
DD5C        LD E,IXH*
DD5D        LD E,IXL*
DD5E d      LD E,(IX+d)
DD60        LD IXH,B*
DD61        LD IXH,C*
DD62        LD IXH,D*
DD63        LD IXH,E*
DD64        LD IXH,IXH*
DD65        LD IXH,IXL*
DD66 d      LD H,(IX+d)
DD67        LD IXH,A*
DD68        LD IXL,B*
DD69        LD IXL,C*
DD6A        LD IXL,D*
DD6B        LD IXL,E*
DD6C        LD IXL,IXH*
DD6D        LD IXL,IXL*
DD6E d      LD L,(IX+d)
DD6F        LD IXL,A*
DD70 d      LD (IX+d),B
DD71 d      LD (IX+d),C
DD72 d      LD (IX+d),D
DD73 d      LD (IX+d),E
DD74 d      LD (IX+d),H
DD75 d      LD (IX+d),L
DD77 d      LD (IX+d),A
DD7C        LD A,IXH*
DD7D        LD A,IXL*
DD7E d      LD A,(IX+d)
DD84        ADD A,IXH*
DD85        ADD A,IXL*
DD86 d      ADD A,(IX+d)
DD8C        ADC A,IXH*
DD8D        ADC A,IXL*
DD8E d      ADC A,(IX+d)
DD94        SUB IXH*
DD95        SUB IXL*
DD96 d      SUB (IX+d)
DD9C        SBC A,IXH*
DD9D        SBC A,IXL*
DD9E d      SBC A,(IX+d)
DDA4        AND IXH*
DDA5        AND IXL*
DDA6 d      AND (IX+d)
DDAC        XOR IXH*
DDAD        XOR IXL*
```

```
DDAE d      XOR (IX+d)
DDB4        OR IXH*
DDB5        OR IXL*
DDB6 d      OR (IX+d)
DDBC        CP IXH*
DDBD        CP IXL*
DDBE d      CP (IX+d)
DDCB d 00   LD B,RLC (IX+d)*
DDCB d 01   LD C,RLC (IX+d)*
DDCB d 02   LD D,RLC (IX+d)*
DDCB d 03   LD E,RLC (IX+d)*
DDCB d 04   LD H,RLC (IX+d)*
DDCB d 05   LD L,RLC (IX+d)*
DDCB d 06   RLC (IX+d)
DDCB d 07   LD A,RLC (IX+d)*
DDCB d 08   LD B,RRC (IX+d)*
DDCB d 09   LD C,RRC (IX+d)*
DDCB d 0A   LD D,RRC (IX+d)*
DDCB d 0B   LD E,RRC (IX+d)*
DDCB d 0C   LD H,RRC (IX+d)*
DDCB d 0D   LD L,RRC (IX+d)*
DDCB d 0E   RRC (IX+d)
DDCB d 0F   LD A,RRC (IX+d)*
DDCB d 10   LD B,RL (IX+d)*
DDCB d 11   LD C,RL (IX+d)*
DDCB d 12   LD D,RL (IX+d)*
DDCB d 13   LD E,RL (IX+d)*
DDCB d 14   LD H,RL (IX+d)*
DDCB d 15   LD L,RL (IX+d)*
DDCB d 16   RL (IX+d)
DDCB d 17   LD A,RL (IX+d)*
DDCB d 18   LD B,RR (IX+d)*
DDCB d 19   LD C,RR (IX+d)*
DDCB d 1A   LD D,RR (IX+d)*
DDCB d 1B   LD E,RR (IX+d)*
DDCB d 1C   LD H,RR (IX+d)*
DDCB d 1D   LD L,RR (IX+d)*
DDCB d 1E   RR (IX+d)
DDCB d 1F   LD A,RR (IX+d)*
DDCB d 20   LD B,SLA (IX+d)*
DDCB d 21   LD C,SLA (IX+d)*
DDCB d 22   LD D,SLA (IX+d)*
DDCB d 23   LD E,SLA (IX+d)*
DDCB d 24   LD H,SLA (IX+d)*
DDCB d 25   LD L,SLA (IX+d)*
DDCB d 26   SLA (IX+d)
DDCB d 27   LD A,SLA (IX+d)*
DDCB d 28   LD B,SRA (IX+d)*
DDCB d 29   LD C,SRA (IX+d)*
DDCB d 2A   LD D,SRA (IX+d)*
DDCB d 2B   LD E,SRA (IX+d)*
DDCB d 2C   LD H,SRA (IX+d)*
DDCB d 2D   LD L,SRA (IX+d)*
DDCB d 2E   SRA (IX+d)
DDCB d 2F   LD A,SRA (IX+d)*
DDCB d 30   LD B,SLL (IX+d)*
DDCB d 31   LD C,SLL (IX+d)*
DDCB d 32   LD D,SLL (IX+d)*
DDCB d 33   LD E,SLL (IX+d)*
DDCB d 34   LD H,SLL (IX+d)*
DDCB d 35   LD L,SLL (IX+d)*
DDCB d 36   SLL (IX+d)*
DDCB d 37   LD A,SLL (IX+d)*
DDCB d 38   LD B,SRL (IX+d)*
DDCB d 39   LD C,SRL (IX+d)*
DDCB d 3A   LD D,SRL (IX+d)*
DDCB d 3B   LD E,SRL (IX+d)*
DDCB d 3C   LD H,SRL (IX+d)*
DDCB d 3D   LD L,SRL (IX+d)*
DDCB d 3E   SRL (IX+d)
DDCB d 3F   LD A,SRL (IX+d)*
DDCB d 40   BIT 0,(IX+d)*
DDCB d 41   BIT 0,(IX+d)*
DDCB d 42   BIT 0,(IX+d)*
DDCB d 43   BIT 0,(IX+d)*
DDCB d 44   BIT 0,(IX+d)*
DDCB d 45   BIT 0,(IX+d)*
DDCB d 46   BIT 0,(IX+d)
DDCB d 47   BIT 0,(IX+d)*
DDCB d 48   BIT 1,(IX+d)*
DDCB d 49   BIT 1,(IX+d)*
DDCB d 4A   BIT 1,(IX+d)*
DDCB d 4B   BIT 1,(IX+d)*
DDCB d 4C   BIT 1,(IX+d)*
DDCB d 4D   BIT 1,(IX+d)*
DDCB d 4E   BIT 1,(IX+d)
DDCB d 4F   BIT 1,(IX+d)*
DDCB d 50   BIT 2,(IX+d)*
DDCB d 51   BIT 2,(IX+d)*
DDCB d 52   BIT 2,(IX+d)*
```

```
DDCB d 53   BIT 2,(IX+d)*
DDCB d 54   BIT 2,(IX+d)*
DDCB d 55   BIT 2,(IX+d)*
DDCB d 56   BIT 2,(IX+d)
DDCB d 57   BIT 2,(IX+d)*
DDCB d 58   BIT 3,(IX+d)*
DDCB d 59   BIT 3,(IX+d)*
DDCB d 5A   BIT 3,(IX+d)*
DDCB d 5B   BIT 3,(IX+d)*
DDCB d 5C   BIT 3,(IX+d)*
DDCB d 5D   BIT 3,(IX+d)*
DDCB d 5E   BIT 3,(IX+d)
DDCB d 5F   BIT 3,(IX+d)*
DDCB d 60   BIT 4,(IX+d)*
DDCB d 61   BIT 4,(IX+d)*
DDCB d 62   BIT 4,(IX+d)*
DDCB d 63   BIT 4,(IX+d)*
DDCB d 64   BIT 4,(IX+d)*
DDCB d 65   BIT 4,(IX+d)*
DDCB d 66   BIT 4,(IX+d)
DDCB d 67   BIT 4,(IX+d)*
DDCB d 68   BIT 5,(IX+d)*
DDCB d 69   BIT 5,(IX+d)*
DDCB d 6A   BIT 5,(IX+d)*
DDCB d 6B   BIT 5,(IX+d)*
DDCB d 6C   BIT 5,(IX+d)*
DDCB d 6D   BIT 5,(IX+d)*
DDCB d 6E   BIT 5,(IX+d)
DDCB d 6F   BIT 5,(IX+d)*
DDCB d 70   BIT 6,(IX+d)*
DDCB d 71   BIT 6,(IX+d)*
DDCB d 72   BIT 6,(IX+d)*
DDCB d 73   BIT 6,(IX+d)*
DDCB d 74   BIT 6,(IX+d)*
DDCB d 75   BIT 6,(IX+d)*
DDCB d 76   BIT 6,(IX+d)
DDCB d 77   BIT 6,(IX+d)*
DDCB d 78   BIT 7,(IX+d)*
DDCB d 79   BIT 7,(IX+d)*
DDCB d 7A   BIT 7,(IX+d)*
DDCB d 7B   BIT 7,(IX+d)*
DDCB d 7C   BIT 7,(IX+d)*
DDCB d 7D   BIT 7,(IX+d)*
DDCB d 7E   BIT 7,(IX+d)
DDCB d 7F   BIT 7,(IX+d)*
```

```
DDCB d 80   LD B,RES 0,(IX+d)*
DDCB d 81   LD C,RES 0,(IX+d)*
DDCB d 82   LD D,RES 0,(IX+d)*
DDCB d 83   LD E,RES 0,(IX+d)*
DDCB d 84   LD H,RES 0,(IX+d)*
DDCB d 85   LD L,RES 0,(IX+d)*
DDCB d 86   RES 0,(IX+d)
DDCB d 87   LD A,RES 0,(IX+d)*
DDCB d 88   LD B,RES 1,(IX+d)*
DDCB d 89   LD C,RES 1,(IX+d)*
DDCB d 8A   LD D,RES 1,(IX+d)*
DDCB d 8B   LD E,RES 1,(IX+d)*
DDCB d 8C   LD H,RES 1,(IX+d)*
DDCB d 8D   LD L,RES 1,(IX+d)*
DDCB d 8E   RES 1,(IX+d)
DDCB d 8F   LD A,RES 1,(IX+d)*
DDCB d 90   LD B,RES 2,(IX+d)*
DDCB d 91   LD C,RES 2,(IX+d)*
DDCB d 92   LD D,RES 2,(IX+d)*
DDCB d 93   LD E,RES 2,(IX+d)*
DDCB d 94   LD H,RES 2,(IX+d)*
DDCB d 95   LD L,RES 2,(IX+d)*
DDCB d 96   RES 2,(IX+d)
DDCB d 97   LD A,RES 2,(IX+d)*
DDCB d 98   LD B,RES 3,(IX+d)*
DDCB d 99   LD C,RES 3,(IX+d)*
DDCB d 9A   LD D,RES 3,(IX+d)*
DDCB d 9B   LD E,RES 3,(IX+d)*
DDCB d 9C   LD H,RES 3,(IX+d)*
DDCB d 9D   LD L,RES 3,(IX+d)*
DDCB d 9E   RES 3,(IX+d)
DDCB d 9F   LD A,RES 3,(IX+d)*
DDCB d A0   LD B,RES 4,(IX+d)*
DDCB d A1   LD C,RES 4,(IX+d)*
DDCB d A2   LD D,RES 4,(IX+d)*
DDCB d A3   LD E,RES 4,(IX+d)*
DDCB d A4   LD H,RES 4,(IX+d)*
DDCB d A5   LD L,RES 4,(IX+d)*
DDCB d A6   RES 4,(IX+d)
DDCB d A7   LD A,RES 4,(IX+d)*
DDCB d A8   LD B,RES 5,(IX+d)*
DDCB d A9   LD C,RES 5,(IX+d)*
DDCB d AA   LD D,RES 5,(IX+d)*
DDCB d AB   LD E,RES 5,(IX+d)*
DDCB d AC   LD H,RES 5,(IX+d)*
```

```
DDCB d AD   LD L,RES 5,(IX+d)*
DDCB d AE   RES 5,(IX+d)
DDCB d AF   LD A,RES 5,(IX+d)*
DDCB d B0   LD B,RES 6,(IX+d)*
DDCB d B1   LD C,RES 6,(IX+d)*
DDCB d B2   LD D,RES 6,(IX+d)*
DDCB d B3   LD E,RES 6,(IX+d)*
DDCB d B4   LD H,RES 6,(IX+d)*
DDCB d B5   LD L,RES 6,(IX+d)*
DDCB d B6   RES 6,(IX+d)
DDCB d B7   LD A,RES 6,(IX+d)*
DDCB d B8   LD B,RES 7,(IX+d)*
DDCB d B9   LD C,RES 7,(IX+d)*
DDCB d BA   LD D,RES 7,(IX+d)*
DDCB d BB   LD E,RES 7,(IX+d)*
DDCB d BC   LD H,RES 7,(IX+d)*
DDCB d BD   LD L,RES 7,(IX+d)*
DDCB d BE   RES 7,(IX+d)
DDCB d BF   LD A,RES 7,(IX+d)*
DDCB d C0   LD B,SET 0,(IX+d)*
DDCB d C1   LD C,SET 0,(IX+d)*
DDCB d C2   LD D,SET 0,(IX+d)*
DDCB d C3   LD E,SET 0,(IX+d)*
DDCB d C4   LD H,SET 0,(IX+d)*
DDCB d C5   LD L,SET 0,(IX+d)*
DDCB d C6   SET 0,(IX+d)
DDCB d C7   LD A,SET 0,(IX+d)*
DDCB d C8   LD B,SET 1,(IX+d)*
DDCB d C9   LD C,SET 1,(IX+d)*
DDCB d CA   LD D,SET 1,(IX+d)*
DDCB d CB   LD E,SET 1,(IX+d)*
DDCB d CC   LD H,SET 1,(IX+d)*
DDCB d CD   LD L,SET 1,(IX+d)*
DDCB d CE   SET 1,(IX+d)
DDCB d CF   LD A,SET 1,(IX+d)*
DDCB d D0   LD B,SET 2,(IX+d)*
DDCB d D1   LD C,SET 2,(IX+d)*
DDCB d D2   LD D,SET 2,(IX+d)*
DDCB d D3   LD E,SET 2,(IX+d)*
DDCB d D4   LD H,SET 2,(IX+d)*
DDCB d D5   LD L,SET 2,(IX+d)*
DDCB d D6   SET 2,(IX+d)
DDCB d D7   LD A,SET 2,(IX+d)*
DDCB d D8   LD B,SET 3,(IX+d)*
DDCB d D9   LD C,SET 3,(IX+d)*
DDCB d DA   LD D,SET 3,(IX+d)*
DDCB d DB   LD E,SET 3,(IX+d)*
DDCB d DC   LD H,SET 3,(IX+d)*
DDCB d DD   LD L,SET 3,(IX+d)*
DDCB d DE   SET 3,(IX+d)
DDCB d DF   LD A,SET 3,(IX+d)*
DDCB d E0   LD B,SET 4,(IX+d)*
DDCB d E1   LD C,SET 4,(IX+d)*
DDCB d E2   LD D,SET 4,(IX+d)*
DDCB d E3   LD E,SET 4,(IX+d)*
DDCB d E4   LD H,SET 4,(IX+d)*
DDCB d E5   LD L,SET 4,(IX+d)*
DDCB d E6   SET 4,(IX+d)
DDCB d E7   LD A,SET 4,(IX+d)*
DDCB d E8   LD B,SET 5,(IX+d)*
DDCB d E9   LD C,SET 5,(IX+d)*
DDCB d EA   LD D,SET 5,(IX+d)*
DDCB d EB   LD E,SET 5,(IX+d)*
DDCB d EC   LD H,SET 5,(IX+d)*
DDCB d ED   LD L,SET 5,(IX+d)*
DDCB d EE   SET 5,(IX+d)
DDCB d EF   LD A,SET 5,(IX+d)*
DDCB d F0   LD B,SET 6,(IX+d)*
DDCB d F1   LD C,SET 6,(IX+d)*
DDCB d F2   LD D,SET 6,(IX+d)*
DDCB d F3   LD E,SET 6,(IX+d)*
DDCB d F4   LD H,SET 6,(IX+d)*
DDCB d F5   LD L,SET 6,(IX+d)*
DDCB d F6   SET 6,(IX+d)
DDCB d F7   LD A,SET 6,(IX+d)*
DDCB d F8   LD B,SET 7,(IX+d)*
DDCB d F9   LD C,SET 7,(IX+d)*
DDCB d FA   LD D,SET 7,(IX+d)*
DDCB d FB   LD E,SET 7,(IX+d)*
DDCB d FC   LD H,SET 7,(IX+d)*
DDCB d FD   LD L,SET 7,(IX+d)*
DDCB d FE   SET 7,(IX+d)
DDCB d FF   LD A,SET 7,(IX+d)*
DDE1        POP IX
DDE3        EX (SP),IX
DDE5        PUSH IX
DDE9        JP IX
DDF9        LD SP,IX
DE n        SBC A,n
DF          RST 18h
```

```
E0          RET PO
E1          POP HL
E2 n n      JP PO,nn
E3          EX (SP),HL
E4 n n      CALL PO,nn
E5          PUSH HL
E6 n        AND n
E7          RST 20h
E8          RET PE
E9          JP HL
EA n n      JP PE,nn
EB          EX DE,HL
EC n n      CALL PE,nn
ED40        IN B,(C)
ED41        OUT (C),B
ED42        SBC HL,BC
ED43 n n    LD (nn),BC
ED44        NEG
ED45        RETN
ED46        IM 0
ED47        LD I,A
ED48        IN C,(C)
ED49        OUT (C),C
ED4A        ADC HL,BC
ED4B n n    LD BC,(nn)
ED4C        NEG*
ED4D        RETI
ED4E        IM 0/1*
ED4F        LD R,A
ED50        IN D,(C)
ED51        OUT (C),D
ED52        SBC HL,DE
ED53 n n    LD (nn),DE
ED54        NEG*
ED55        RETI/N*
ED56        IM 1
ED57        LD A,I
ED58        IN E,(C)
ED59        OUT (C),E
ED5A        ADC HL,DE
ED5B n n    LD DE,(nn)
ED5C        NEG*
ED5D        RETI/N*
ED5E        IM 2
ED5F        LD A,R
ED60        IN H,(C)
ED61        OUT (C),H
ED62        SBC HL,HL
ED63 n n    LD (nn),HL
ED64        NEG*
ED65        RETI/N*
ED66        IM 0*
ED67        RRD
ED68        IN L,(C)
ED69        OUT (C),L
ED6A        ADC HL,HL
ED6B n n    LD HL,(nn)
ED6C        NEG*
ED6D        RETI/N*
ED6E        IM 0/1*
ED6F        RLD
ED70        IN (C)* / IN F,(C)*
ED71        OUT (C),0*
ED72        SBC HL,SP
ED73 n n    LD (nn),SP
ED74        NEG*
ED75        RETI/N*
ED76        IM 1*
ED78        IN A,(C)
ED79        OUT (C),A
ED7A        ADC HL,SP
ED7B n n    LD SP,(nn)
ED7C        NEG*
ED7D        RETI/N*
ED7E        IM 2*
EDA0        LDI
EDA1        CPI
EDA2        INI
EDA3        OUTI
EDA8        LDD
EDA9        CPD
EDAA        IND
EDAB        OUTD
EDB0        LDIR
EDB1        CPIR
EDB2        INIR
EDB3        OTIR
EDB8        LDDR
EDB9        CPDR
EDBA        INDR
```

```
EDBB        OTDR
EE n        XOR n
EF          RST 28h
F0          RET P
F1          POP AF
F2 n n      JP P,nn
F3          DI
F4 n n      CALL P,nn
F5          PUSH AF
F6 n        OR n
F7          RST 30h
F8          RET M
F9          LD SP,HL
FA n n      JP M,nn
FB          EI
FC n n      CALL M,nn
FD09        ADD IY,BC
FD19        ADD IY,DE
FD21 n n    LD IY,nn
FD22 n n    LD (nn),IY
FD23        INC IY
FD24        INC IYH*
FD25        DEC IYH*
FD26 n      LD IYH,n*
FD29        ADD IY,IY
FD2A n n    LD IY,(nn)
FD2B        DEC IY
FD2C        INC IYL*
FD2D        DEC IYL*
FD2E n      LD IYL,n*
FD34 d      INC (IY+d)
FD35 d      DEC (IY+d)
FD36 d n    LD (IY+d),n
FD39        ADD IY,SP
FD44        LD B,IYH*
FD45        LD B,IYL*
FD46 d      LD B,(IY+d)
FD4C        LD C,IYH*
FD4D        LD C,IYL*
FD4E d      LD C,(IY+d)
FD54        LD D,IYH*
FD55        LD D,IYL*
FD56 d      LD D,(IY+d)
FD5C        LD E,IYH*
FD5D        LD E,IYL*
FD5E d      LD E,(IY+d)
FD60        LD IYH,B*
FD61        LD IYH,C*
FD62        LD IYH,D*
FD63        LD IYH,E*
FD64        LD IYH,IYH*
FD65        LD IYH,IYL*
FD66 d      LD H,(IY+d)
FD67        LD IYH,A*
FD68        LD IYL,B*
FD69        LD IYL,C*
FD6A        LD IYL,D*
FD6B        LD IYL,E*
FD6C        LD IYL,IYH*
FD6D        LD IYL,IYL*
FD6E d      LD L,(IY+d)
FD6F        LD IYL,A*
FD70 d      LD (IY+d),B
FD71 d      LD (IY+d),C
FD72 d      LD (IY+d),D
FD73 d      LD (IY+d),E
FD74 d      LD (IY+d),H
FD75 d      LD (IY+d),L
FD77 d      LD (IY+d),A
FD7C        LD A,IYH*
FD7D        LD A,IYL*
FD7E d      LD A,(IY+d)
FD84        ADD A,IYH*
FD85        ADD A,IYL*
FD86 d      ADD A,(IY+d)
FD8C        ADC A,IYH*
FD8D        ADC A,IYL*
FD8E d      ADC A,(IY+d)
FD94        SUB IYH*
FD95        SUB IYL*
FD96 d      SUB (IY+d)
FD9C        SBC A,IYH*
FD9D        SBC A,IYL*
FD9E d      SBC A,(IY+d)
FDA4        AND IYH*
FDA5        AND IYL*
FDA6 d      AND (IY+d)
FDAC        XOR IYH*
FDAD        XOR IYL*
FDAE d      XOR (IY+d)
```

```
FDB4        OR IYH*
FDB5        OR IYL*
FDB6 d      OR (IY+d)
FDBC        CP IYH*
FDBD        CP IYL*
FDBE d      CP (IY+d)
FDCB d 00   LD B,RLC (IY+d)*
FDCB d 01   LD C,RLC (IY+d)*
FDCB d 02   LD D,RLC (IY+d)*
FDCB d 03   LD E,RLC (IY+d)*
FDCB d 04   LD H,RLC (IY+d)*
FDCB d 05   LD L,RLC (IY+d)*
FDCB d 06   RLC (IY+d)
FDCB d 07   LD A,RLC (IY+d)*
FDCB d 08   LD B,RRC (IY+d)*
FDCB d 09   LD C,RRC (IY+d)*
FDCB d 0A   LD D,RRC (IY+d)*
FDCB d 0B   LD E,RRC (IY+d)*
FDCB d 0C   LD H,RRC (IY+d)*
FDCB d 0D   LD L,RRC (IY+d)*
FDCB d 0E   RRC (IY+d)
FDCB d 0F   LD A,RRC (IY+d)*
FDCB d 10   LD B,RL (IY+d)*
FDCB d 11   LD C,RL (IY+d)*
FDCB d 12   LD D,RL (IY+d)*
FDCB d 13   LD E,RL (IY+d)*
FDCB d 14   LD H,RL (IY+d)*
FDCB d 15   LD L,RL (IY+d)*
FDCB d 16   RL (IY+d)
FDCB d 17   LD A,RL (IY+d)*
FDCB d 18   LD B,RR (IY+d)*
FDCB d 19   LD C,RR (IY+d)*
FDCB d 1A   LD D,RR (IY+d)*
FDCB d 1B   LD E,RR (IY+d)*
FDCB d 1C   LD H,RR (IY+d)*
FDCB d 1D   LD L,RR (IY+d)*
FDCB d 1E   RR (IY+d)
FDCB d 1F   LD A,RR (IY+d)*
FDCB d 20   LD B,SLA (IY+d)*
FDCB d 21   LD C,SLA (IY+d)*
FDCB d 22   LD D,SLA (IY+d)*
FDCB d 23   LD E,SLA (IY+d)*
FDCB d 24   LD H,SLA (IY+d)*
FDCB d 25   LD L,SLA (IY+d)*
FDCB d 26   SLA (IY+d)
FDCB d 27   LD A,SLA (IY+d)*
FDCB d 28   LD B,SRA (IY+d)*
FDCB d 29   LD C,SRA (IY+d)*
FDCB d 2A   LD D,SRA (IY+d)*
FDCB d 2B   LD E,SRA (IY+d)*
FDCB d 2C   LD H,SRA (IY+d)*
FDCB d 2D   LD L,SRA (IY+d)*
FDCB d 2E   SRA (IY+d)
FDCB d 2F   LD A,SRA (IY+d)*
FDCB d 30   LD B,SLL (IY+d)*
FDCB d 31   LD C,SLL (IY+d)*
FDCB d 32   LD D,SLL (IY+d)*
FDCB d 33   LD E,SLL (IY+d)*
FDCB d 34   LD H,SLL (IY+d)*
FDCB d 35   LD L,SLL (IY+d)*
FDCB d 36   SLL (IY+d)*
FDCB d 37   LD A,SLL (IY+d)*
FDCB d 38   LD B,SRL (IY+d)*
FDCB d 39   LD C,SRL (IY+d)*
FDCB d 3A   LD D,SRL (IY+d)*
FDCB d 3B   LD E,SRL (IY+d)*
FDCB d 3C   LD H,SRL (IY+d)*
FDCB d 3D   LD L,SRL (IY+d)*
FDCB d 3E   SRL (IY+d)
FDCB d 3F   LD A,SRL (IY+d)*
FDCB d 40   BIT 0,(IY+d)*
FDCB d 41   BIT 0,(IY+d)*
FDCB d 42   BIT 0,(IY+d)*
FDCB d 43   BIT 0,(IY+d)*
FDCB d 44   BIT 0,(IY+d)*
FDCB d 45   BIT 0,(IY+d)*
FDCB d 46   BIT 0,(IY+d)
FDCB d 47   BIT 0,(IY+d)*
FDCB d 48   BIT 1,(IY+d)*
FDCB d 49   BIT 1,(IY+d)*
FDCB d 4A   BIT 1,(IY+d)*
FDCB d 4B   BIT 1,(IY+d)*
FDCB d 4C   BIT 1,(IY+d)*
FDCB d 4D   BIT 1,(IY+d)*
FDCB d 4E   BIT 1,(IY+d)
FDCB d 4F   BIT 1,(IY+d)*
FDCB d 50   BIT 2,(IY+d)*
FDCB d 51   BIT 2,(IY+d)*
FDCB d 52   BIT 2,(IY+d)*
FDCB d 53   BIT 2,(IY+d)*
```

```
FDCB d 54   BIT 2,(IY+d)*
FDCB d 55   BIT 2,(IY+d)*
FDCB d 56   BIT 2,(IY+d)
FDCB d 57   BIT 2,(IY+d)*
FDCB d 58   BIT 3,(IY+d)*
FDCB d 59   BIT 3,(IY+d)*
FDCB d 5A   BIT 3,(IY+d)*
FDCB d 5B   BIT 3,(IY+d)*
FDCB d 5C   BIT 3,(IY+d)*
FDCB d 5D   BIT 3,(IY+d)*
FDCB d 5E   BIT 3,(IY+d)
FDCB d 5F   BIT 3,(IY+d)*
FDCB d 60   BIT 4,(IY+d)*
FDCB d 61   BIT 4,(IY+d)*
FDCB d 62   BIT 4,(IY+d)*
FDCB d 63   BIT 4,(IY+d)*
FDCB d 64   BIT 4,(IY+d)*
FDCB d 65   BIT 4,(IY+d)*
FDCB d 66   BIT 4,(IY+d)
FDCB d 67   BIT 4,(IY+d)*
FDCB d 68   BIT 5,(IY+d)*
FDCB d 69   BIT 5,(IY+d)*
FDCB d 6A   BIT 5,(IY+d)*
FDCB d 6B   BIT 5,(IY+d)*
FDCB d 6C   BIT 5,(IY+d)*
FDCB d 6D   BIT 5,(IY+d)*
FDCB d 6E   BIT 5,(IY+d)
FDCB d 6F   BIT 5,(IY+d)*
FDCB d 70   BIT 6,(IY+d)*
FDCB d 71   BIT 6,(IY+d)*
FDCB d 72   BIT 6,(IY+d)*
FDCB d 73   BIT 6,(IY+d)*
FDCB d 74   BIT 6,(IY+d)*
FDCB d 75   BIT 6,(IY+d)*
FDCB d 76   BIT 6,(IY+d)
FDCB d 77   BIT 6,(IY+d)*
FDCB d 78   BIT 7,(IY+d)*
FDCB d 79   BIT 7,(IY+d)*
FDCB d 7A   BIT 7,(IY+d)*
FDCB d 7B   BIT 7,(IY+d)*
FDCB d 7C   BIT 7,(IY+d)*
FDCB d 7D   BIT 7,(IY+d)*
FDCB d 7E   BIT 7,(IY+d)
FDCB d 7F   BIT 7,(IY+d)*
FDCB d 80   LD B,RES 0,(IY+d)*
FDCB d 81   LD C,RES 0,(IY+d)*
FDCB d 82   LD D,RES 0,(IY+d)*
FDCB d 83   LD E,RES 0,(IY+d)*
FDCB d 84   LD H,RES 0,(IY+d)*
FDCB d 85   LD L,RES 0,(IY+d)*
FDCB d 86   RES 0,(IY+d)
FDCB d 87   LD A,RES 0,(IY+d)*
FDCB d 88   LD B,RES 1,(IY+d)*
FDCB d 89   LD C,RES 1,(IY+d)*
FDCB d 8A   LD D,RES 1,(IY+d)*
FDCB d 8B   LD E,RES 1,(IY+d)*
FDCB d 8C   LD H,RES 1,(IY+d)*
FDCB d 8D   LD L,RES 1,(IY+d)*
FDCB d 8E   RES 1,(IY+d)
FDCB d 8F   LD A,RES 1,(IY+d)*
FDCB d 90   LD B,RES 2,(IY+d)*
FDCB d 91   LD C,RES 2,(IY+d)*
FDCB d 92   LD D,RES 2,(IY+d)*
FDCB d 93   LD E,RES 2,(IY+d)*
FDCB d 94   LD H,RES 2,(IY+d)*
FDCB d 95   LD L,RES 2,(IY+d)*
FDCB d 96   RES 2,(IY+d)
FDCB d 97   LD A,RES 2,(IY+d)*
FDCB d 98   LD B,RES 3,(IY+d)*
FDCB d 99   LD C,RES 3,(IY+d)*
FDCB d 9A   LD D,RES 3,(IY+d)*
FDCB d 9B   LD E,RES 3,(IY+d)*
FDCB d 9C   LD H,RES 3,(IY+d)*
FDCB d 9D   LD L,RES 3,(IY+d)*
FDCB d 9E   RES 3,(IY+d)
FDCB d 9F   LD A,RES 3,(IY+d)*
FDCB d A0   LD B,RES 4,(IY+d)*
FDCB d A1   LD C,RES 4,(IY+d)*
FDCB d A2   LD D,RES 4,(IY+d)*
FDCB d A3   LD E,RES 4,(IY+d)*
FDCB d A4   LD H,RES 4,(IY+d)*
FDCB d A5   LD L,RES 4,(IY+d)*
FDCB d A6   RES 4,(IY+d)
FDCB d A7   LD A,RES 4,(IY+d)*
FDCB d A8   LD B,RES 5,(IY+d)*
FDCB d A9   LD C,RES 5,(IY+d)*
FDCB d AA   LD D,RES 5,(IY+d)*
FDCB d AB   LD E,RES 5,(IY+d)*
FDCB d AC   LD H,RES 5,(IY+d)*
FDCB d AD   LD L,RES 5,(IY+d)*
```

```
FDCB d AE   RES 5,(IY+d)
FDCB d AF   LD A,RES 5,(IY+d)*
FDCB d B0   LD B,RES 6,(IY+d)*
FDCB d B1   LD C,RES 6,(IY+d)*
FDCB d B2   LD D,RES 6,(IY+d)*
FDCB d B3   LD E,RES 6,(IY+d)*
FDCB d B4   LD H,RES 6,(IY+d)*
FDCB d B5   LD L,RES 6,(IY+d)*
FDCB d B6   RES 6,(IY+d)
FDCB d B7   LD A,RES 6,(IY+d)*
FDCB d B8   LD B,RES 7,(IY+d)*
FDCB d B9   LD C,RES 7,(IY+d)*
FDCB d BA   LD D,RES 7,(IY+d)*
FDCB d BB   LD E,RES 7,(IY+d)*
FDCB d BC   LD H,RES 7,(IY+d)*
FDCB d BD   LD L,RES 7,(IY+d)*
FDCB d BE   RES 7,(IY+d)
FDCB d BF   LD A,RES 7,(IY+d)*
FDCB d C0   LD B,SET 0,(IY+d)*
FDCB d C1   LD C,SET 0,(IY+d)*
FDCB d C2   LD D,SET 0,(IY+d)*
FDCB d C3   LD E,SET 0,(IY+d)*
FDCB d C4   LD H,SET 0,(IY+d)*
FDCB d C5   LD L,SET 0,(IY+d)*
FDCB d C6   SET 0,(IY+d)
FDCB d C7   LD A,SET 0,(IY+d)*
FDCB d C8   LD B,SET 1,(IY+d)*
FDCB d C9   LD C,SET 1,(IY+d)*
FDCB d CA   LD D,SET 1,(IY+d)*
FDCB d CB   LD E,SET 1,(IY+d)*
FDCB d CC   LD H,SET 1,(IY+d)*
FDCB d CD   LD L,SET 1,(IY+d)*
FDCB d CE   SET 1,(IY+d)
FDCB d CF   LD A,SET 1,(IY+d)*
FDCB d D0   LD B,SET 2,(IY+d)*
FDCB d D1   LD C,SET 2,(IY+d)*
FDCB d D2   LD D,SET 2,(IY+d)*
FDCB d D3   LD E,SET 2,(IY+d)*
FDCB d D4   LD H,SET 2,(IY+d)*
FDCB d D5   LD L,SET 2,(IY+d)*
FDCB d D6   SET 2,(IY+d)
FDCB d D7   LD A,SET 2,(IY+d)*
FDCB d D8   LD B,SET 3,(IY+d)*
FDCB d D9   LD C,SET 3,(IY+d)*
FDCB d DA   LD D,SET 3,(IY+d)*
FDCB d DB   LD E,SET 3,(IY+d)*
FDCB d DC   LD H,SET 3,(IY+d)*
FDCB d DD   LD L,SET 3,(IY+d)*
FDCB d DE   SET 3,(IY+d)
FDCB d DF   LD A,SET 3,(IY+d)*
FDCB d E0   LD B,SET 4,(IY+d)*
FDCB d E1   LD C,SET 4,(IY+d)*
FDCB d E2   LD D,SET 4,(IY+d)*
FDCB d E3   LD E,SET 4,(IY+d)*
FDCB d E4   LD H,SET 4,(IY+d)*
FDCB d E5   LD L,SET 4,(IY+d)*
FDCB d E6   SET 4,(IY+d)
FDCB d E7   LD A,SET 4,(IY+d)*
FDCB d E8   LD B,SET 5,(IY+d)*
FDCB d E9   LD C,SET 5,(IY+d)*
FDCB d EA   LD D,SET 5,(IY+d)*
FDCB d EB   LD E,SET 5,(IY+d)*
FDCB d EC   LD H,SET 5,(IY+d)*
FDCB d ED   LD L,SET 5,(IY+d)*
FDCB d EE   SET 5,(IY+d)
FDCB d EF   LD A,SET 5,(IY+d)*
FDCB d F0   LD B,SET 6,(IY+d)*
FDCB d F1   LD C,SET 6,(IY+d)*
FDCB d F2   LD D,SET 6,(IY+d)*
FDCB d F3   LD E,SET 6,(IY+d)*
FDCB d F4   LD H,SET 6,(IY+d)*
FDCB d F5   LD L,SET 6,(IY+d)*
FDCB d F6   SET 6,(IY+d)
FDCB d F7   LD A,SET 6,(IY+d)*
FDCB d F8   LD B,SET 7,(IY+d)*
FDCB d F9   LD C,SET 7,(IY+d)*
FDCB d FA   LD D,SET 7,(IY+d)*
FDCB d FB   LD E,SET 7,(IY+d)*
FDCB d FC   LD H,SET 7,(IY+d)*
FDCB d FD   LD L,SET 7,(IY+d)*
FDCB d FE   SET 7,(IY+d)
FDCB d FF   LD A,SET 7,(IY+d)*
FDE1        POP IY
FDE3        EX (SP),IY
FDE5        PUSH IY
FDE9        JP IY
FDF9        LD SP,IY
FE n        CP n
FF          RST 38h
```

6. TASM Documentation

6.1. About TASM

The Telemark Assembler (TASM) User's Manual

Version 3.1
February, 1998

Thomas N. Anderson
Squak Valley Software
837 Front Street South,
Issaquah, WA 98027
Compuserve: 73770,3612

http://www.halcyon.com/squakvly/

6.2. Table of contents

6.3. Introduction

The Telemark Assembler (TASM) is a table driven cross assembler for the MS-DOS and LINUX environments. Assembly source code, written in the appropriate dialect (generally very close to the manufacturers assembly language), can be assembled with TASM, and the resulting object code transferred to the target microprocessor system via PROM or other mechanisms.

The microprocessor families supported by TASM are:

6502
6800/6801/68HC11
6805
8048
8051
8080/8085, **Z80**
TMS32010, TMS320C25
TMS7000
8096/80196
The user so inclined may build tables for other microprocessors. The descriptions of the various existing tables and instructions on building new tables are not in this document but can be found in the TASMTABS.HTM file on the TASM distribution disk.

TASM characteristics include:

1. Powerful expression parsing (17 operators).
2. Supports a subset of the 'C' preprocessor commands.
3. Macro capability (through use of DEFINE directive).

4. Multiple statements per line.
5. Four object file formats: Intel hex, MOS Technology hex, Motorola hex, binary.
6. Absolute code generation only.
7. Source code available (in C).
8. Uniform syntax across versions for different target machines.
9. Features in support of PROM programming (preset memory, contiguous block).
10. Supports extended instructions for many of the supported microprocessor families.
11. Tables read at run time - single TASM executable for all table versions.
11. Symbol table export for inclusion in subsequent assemblies.
12. Symbol table export file for import with some simulator products.

6.4. Shareware

TASM is distributed as shareware. TASM is not in the public domain. The TASM distribution files may be freely copied (excluding the source code files) and freely used for the purpose of evaluating the suitability of TASM for a given purpose. Use of TASM beyond a reasonable evaluation period requires registration. Prolonged use without registration is unethical.

TASM. Copyright (C) 1998 by Squak Valley Software.
All rights reserved.

6.5. Invocation

TASM can be invoked as follows (optional fields shown in brackets, symbolic fields in italics):

```
tasm -pn [-options ...] src_file [obj_file [lst_file [exp_file
[sym_file]]]]
```

Where options can be one or more of the following:

-table	Specify version (table = table designation)
-ttable	Table (alternate form of above)
-aamask	Assembly control (optional error checking)
-b	Produce object in binary (.COM) format
-c	Object file written as a contiguous block
-dmacro	Define a macro (or just a macro label)
-e	Show source lines after macro expansion
-ffillbyte	Fill entire memory space with fillbyte (hex)
-gobjtype	Object file (0=Intel Hex, 1=MOS Tech, 2=Motorola, 3=binary,4=Intel Hex (Word))
-h	Produce hex table of the assembled code (in list file)
-i	Ignore case for labels
-l[al]	Produce a label table in the listing
-m	Produce object in MOS Technology format
-oobytes	Bytes per object record (for hex obj formats)
-p[lines]	Page the listing file (lines per page. default=60)
-q	Quiet, disable the listing file
-rkb	Set read buffer size in Kbytes (default 2 Kbytes)
-s	Write a symbol table file
-x[xmask]	Enable extended instruction set (if any)
-y	Time the assembly

The filename parameters are defined as follows:

src_file	Source file name
obj_file	Object code file name
lst_file	Listing file name
exp_file	Symbol export file (only if the EXPORT directive is used).
sym_file	Symbol table file (only if the -s option or the SYM/AVSYM directives are used).

The source file must be specified. If not, some usage information is displayed. Default file names for all the other files are generated if they are not explicitly provided. The filename is formed by taking the source filename and changing the extension to one of the following:

Extension File type

.OBJ Object file
.LST Listing file
.EXP Symbol export file
.SYM Symbol table file

TASM has no built-in instruction set tables. Instruction set definition files are read at run time. TASM determines which table to use based on the '-table' field shown above. For example, to assemble the code in a file called source.asm, one would enter

```
tasm -48 source.asm   for an 8048 assembly
tasm -65 source.asm   for a 6502 assembly
tasm -51 source.asm   for an 8051 assembly.
tasm -85 source.asm   for an 8085 assembly.
tasm -80 source.asm   for a Z80 assembly.
tasm -05 source.asm   for a 6805 assembly.
tasm -68 source.asm   for a 6800/6801/68HC11 assembly.
tasm -70 source.asm   for a TMS7000 assembly.
tasm -3210 source.asm    for a TMS32010 assembly.
tasm -3225 source.asm    for a TMS320C25 assembly.
tasm -96 source.asm   for a 8096/80196 assembly
```

Tables are read from a file named by taking the digits specified after the '-' and appending it to 'TASM' then appending the '.TAB' extension. Thus, the -48 flag would cause the tables to be read from the file 'TASM48.TAB'.

It is possible to designate tables by non numeric part numbers if the -t flag is used. For example, if a user built a table called TASMF8.TAB then TASM could be invoked as follows:

```
tasm -tf8 source.asm
```

Each option flag must be preceded by a dash. Options need not precede the file names. The various options are described in the sections that follow.

a - Assembly Control

TASM can provide additional error checking by specifying the -a option at the time of execution. If the -a is provided without a digit following, then all the available error checking is done. If a digit follows, then it is used as a mask to determine the error checks to be made. The bits of the mask are defined as follows:

Bit	Option	Default	Description
0	-a1	OFF	Check for apparent illegal use of indirection
1	-a2	ON	Check for unused data in the arguments
2	-a4	ON	Check for duplicate labels
3	-a8	OFF	Check for non-unary operators at start of expression.

Combinations of the above bits can also be used. For example, -a5 would enable the checking for illegal indirection and duplicate labels.

Illegal indirection applies to micros that use parenthesis around an argument to indicate indirection. Since it is always legal to put an extra pair of parenthesis around any expression (as far as the expression parser is concerned), the user may think that he/she is indicating indirection for an instruction that has no indirection and TASM would not complain. Enabling this checking will result in an error message (warning) whenever an outer pair of parenthesis is used and the instruction set definition table does not explicitly indicate that to be a valid form of addressing.

Unused data in arguments applies to cases where a single byte of data is needed from an argument, but the argument contains more than one byte of data. If a full sixteen bit address is used in a 'Load Immediate' type instruction that needs only a single byte, for example, an error message would be generated. Here is an example (6502 code):

```
0001   1234                     .org $1234
test.asm line 0002: Unused data in MS byte of argument.
0002   1234 A9 34       start  lda  #start
```

To make the above checks occur whenever you do an assembly, add a line like this to your AUTOEXEC.BAT file:

```
SET TASMOPTS=-a
```

b - Binary Object Format

This option causes the object file to be written in binary - one byte for each byte of code/data. Note that no address information is included in the object file in this format. The contiguous block (-c) output mode is forced when this option is invoked. This flag is equivalent to -g3.

c - Contiguous Block Output

If this option is specified, then all bytes in the range from the lowest used byte to the highest will be defined in the object file. Normally, with the default Intel Hex object format enabled, if the Program Counter (PC) jumps forward because of an .ORG directive, the bytes skipped over will not have any value assigned them in the object file. With this option enabled, no output to the object file occurs until the end of the assembly at which time the whole block is written. This is useful when using TASM to generate code that will be put into a PROM so that all locations will have a known value. This option is often used in conjunction with the -f option to ensure all unused bytes will have a known value.

d - Define a Macro

Macros are defined on the command line generally to control the assembly of various IFDEF's that are in the source file. This is a convenient way to generate various versions of object code from a single source file.

e - Expand Source

Normally TASM shows lines in the listing file just as they are in the source file. If macros are in use (via the DEFINE directive) it is sometimes desirable to see the source lines after expansion. Use the '-e' flag to accomplish this.

f - Fill Memory

This option causes the memory image that TASM maintains to be initialized to the value specified by the two hex characters immediately following the 'f'. TASM maintains a memory image that is a full 64K bytes in size (even if the target processor cannot utilize that memory space). Invocation of this option introduces a delay at start up of up to 2 seconds (time required to initialize all 64K bytes).

g - Object File Format

TASM can generate object code in four different formats as indicated below:

Option	Description
-g0	Intel hex (default)
-g1	MOS Technology hex (same as -m)
-g2	Motorola hex
-g3	binary (same as -b)
-g4	Intel hex with word addresses

The -m and -b flags may also be used, as indicated above. If both are used the right-most option on the command line will be obeyed.

See the section on OBJECT FILE FORMATS for descriptions of each of the above.

h - Hex Object Code Table

This option causes a hex table of the produced object code to appear in the listing file. Each line of the table shows sixteen bytes of code.

i - Ignore Case in Labels

TASM is normally case sensitive when dealing with labels. For those that prefer case insensitivity, the '-i' command line option can be employed.

l - Label Table

This option causes a label table to appear in the listing file. Each label is shown with its corresponding value. Macro labels (as established via the DEFINE directives) do not appear.

Two optional suffixes may follow the -l option:

Suffix	Description
l	Use long form listing
a	Show all labels (including local labels)

The suffix should immediately follow the '-l'. Here are some examples:

-l to show non-local labels in the short form
-la to show all labels in the short form
-ll to show non-local labels in the long form
-lal to show all labels in the long form

m - MOS Technology Object Format

This option causes the object file to be written in MOS Technology hex format rather than the default Intel hex format. See section on OBJECT FILE FORMATS for a description of the format.

o - Set Number of Bytes per Object Record

When generating object code in either the MOS Technology format or the Intel hex format, a default of 24 (decimal) bytes of object are defined on each record. This can be altered by invoking the '-o' option immediately followed by two hex digits defining the number of bytes per record desired. For example, if 32 bytes per record are desired, one might invoke TASM as:

```
tasm -48 -o20 source.asm
```

p - Page Listing File

This option causes the listing file to have top of page headers and form feeds inserted at appropriate intervals (every sixty lines of output). To override the default of sixty lines per page, indicate the desired number of lines per page as a decimal number immediately following the '-p'. Here is an example:

```
tasm -48 -p56 source.asm
```

q - Disable Listing File

This option causes all output to the listing file to be suppressed, unless a .LIST directive is encountered in the source file (see LIST/NOLIST directives).

r - Set Read Buffer Size

This option overrides the default read buffer size of 2 Kbytes. The first hexadecimal digit immediately after the 'r' is taken as the number of K bytes to allocate for the read buffer (.e.g. -r8 indicates an 8K byte buffer, -rf indicates a 15K byte buffer). Note that that read buffers are taken from the same memory pool as labels and macro storage, and that additional read buffers are needed if "includes" are used. Thus, using 8K byte buffers may be suitable for most assemblies, but programs with large numbers of symbols may not allow such a value. Also, reducing the buffer size to 1 Kbyte can increase the memory pool available for label storage, if such is needed.

s - Enable Symbol File Generation

If this flag is set, a symbol file is generated at the end of the assembly. The format of the file is one line per label, each label starts in the first column and is followed by white space and then four hexadecimal digits representing the value of the label. The following illustrates the format:

```
label1          FFFE
label2          FFFF
label3          1000
```

The symbol file name can be provided as the fifth file name on the command line, or the name will be generated from the source file name with a '.SYM' extension. The symbol table file can also be generated by invoking the SYM directive. The AVSYM directive also generates the symbol file but in a different format (see section on ASSEMBLER DIRECTIVES).

t - Table Name

As an alternative to specifying the instruction set table as two decimal digits, the table indication may be proceeded by the '-t' option. This is useful if the desired table name starts with a non-numeric. Thus, a table for an F8 might be selected as:

```
tasm -tf8 source.asm
```

TASM would expect to read the instruction set definition tables from a file named TASMF8.TAB.

x - Enable Extended Instruction Set

If a processor family has instructions that are valid for only certain members, this option can be used to enable those beyond the basic standard instruction set. A hex digit may follow the 'x' to indicate a mask value used in selecting the appropriate instruction set. Bit 0 of the mask selects the basic instruction set, thus a '-x1' would have no effect. A '-x3' would enable the basic set plus whatever instructions have bit 1 set in their class mask. A '-x' without a digit following is equivalent to a '-xf' which sets all four of the mask bits. The following table indicates the current extended instruction sets available in the TASM tables:

Base Table	Base Family	Ext 1 (-x3)	Ext 2 (-x7)	Ext 3 (-x5)	Ext 4 (-x9)
48	8048	8041A	8022	8021	
65	6502	R65C02		R65C00/21	
05	6805	M146805 CMOS		HC05C4	
80	Z80	HD64180			
68	**6800**	**6801/6803**	**68HC11**		
51	8051				
85	8080				
3210	TMS32010				
3225	TMS320C25		TMS320C26		
70	TMS7000				

The above table does not attempt to show the many microprocessor family members that may apply under a given column.

See the TASMTABS.TXT on-line document for details on each specific table.

y - Enable Assembly Timing

If this option is enabled TASM will generate a statement of elapsed time and assembled lines per second at the end of the assembly.

6.6. Environment variables

The TASM environment can be customized by using the environment variables listed below:

TASMTABS

The TASMTABS variable specifies the path to be searched for TASM instruction set definition tables. If it is not defined then the table(s) must exist in the current working directory. The following examples illustrate possible usage:

For MSDOS `set TASMTABS=C:\TASM`

For LINUX `TASMTABS=/tasm`

TASMOPTS

This variable specifies TASM command line options that are to be invoked every time TASM is executed. For example, if TASM is being used for 8048 assemblies with binary object file output desired, the following statement would be appropriate in the AUTOEXEC.BAT file:

```
set TASMOPTS=-48 -b
```

TASM. Copyright (C) 1998 by Squak Valley Software.
All rights reserved.

6.7. Exit Codes

When TASM terminates, it will return to the OS the following exit codes:

Exit Code	Definition
0	Normal completion, no assembly errors
1	Normal completion, with assembly errors
2	Abnormal completion, insufficient memory
3	Abnormal completion, file access error
4	Abnormal completion, general error

Exit codes 2 and above will also be accompanied by messages to the console concerning the error.

6.8. Source file format

Statements in the source file must conform to a format as follows (except for assembler directive statements which are described in a subsequent section):
label operation operand comment
All of the fields are optional, under appropriate circumstances. An arbitrary amount of white space (space and tabs) can separate each field (as long as the maximum line length of 255 characters is not exceeded). Each of the fields are described in the following sections.

Label Field

If the first character of the line is alphabetic, it is assumed to be the start of a label. Subsequent characters are accepted as part of that label until a space, tab, or ':' is encountered. The assembler assigns a value to the label corresponding to the current location counter. Labels can be a maximum of 32 characters long. Labels can contain upper and lower case letters, digits, underscores, and periods (the first character must be alphabetic). Labels are case sensitive - the label 'START' is a different label from 'start' - unless the '-i' (ignore case) option is enabled.

Operation Field

The operation field contains an instruction mnemonic which specifies the action to be carried out by the target processor when this instruction is executed. The interpretation of each mnemonic is dependent on the target microprocessor (as indicated by the selected TASM table). The operation field may begin in any column except the first. The operation field is case insensitive.

Operand Field

The operand field specifies the data to be operated on by the instruction. It may include expressions and/or special symbols describing the addressing mode to be used. The actual format and interpretation is dependent on the target processor. For a description of the format for currently supported processors, see the TASMTABS.DOC file on the TASM distribution disk.

Comment Field

The comment field always begins with a semicolon. The rest of the line from the semicolon to the end of the line is ignored by TASM, but passed on to the listing file for annotation purposes. The comment field must be the last field on a line, but it may be the only field, starting in column one, if desired.

Multiple Statement Lines

If the backslash character is encountered on a source line, it is treated as a newline. The remainder of the line following the backslash will be processed as an independent line of source code. This allows one to put multiple statements on a line. This facility is not so useful of itself, but when coupled with the capability of the DEFINE directive, powerful multiple statement macros can be constructed (see section on ASSEMBLER DIRECTIVES). Note that when using the statement separator, the character immediately following it should be considered the first character of a new line, and thus must either be a start of a label or white space (not an instruction). As the examples show, a space is put between the backslash and the start of the next instruction.

Sample Source Listing

Some examples of valid source statements follow (6502 mnemonics shown):

```
lab1        lda     byte1   ;get the first byte
            dec     byte1
            jne     label1
;
lab2        sta     byte2,X
;  a multiple statement line follows
            lda     byte1\ sta byte1+4\ lda byte2\ sta byte2+4
```

6.9. Expressions

Expressions are made up of various syntactic elements combined according to a set of syntactical rules. Expressions can be comprised of the following elements:

- Labels
- Constants
- Location Counter Symbol
- Operators
- Parenthesis

Labels

Labels are strings of characters that have a numeric value associated with them, generally representing an address. Labels can contain upper and lower case letters, digits, underscores, and periods. The first character must be a letter or the local label prefix (default '_'). The value of a label is limited to 32 bit precision. Labels can contain up to 32 characters, all of which are significant (none are ignored when looking at a label's value, as in some assemblers). Case is significant unless the '-i' command line option is invoked.
Local labels must only be unique within the scope of the current module. Modules are defined with the MODULE directive. Here is an example:

```
        .MODULE xxx
        lda regx
        jne _skip
        dec
_skip   rts

        .MODULE yyy
        lda regy
        jne _skip
        dec
_skip   rts
```

In the above example, the *_skip* label is reused without harm. As a default, local labels are not shown in the label table listing (resulting from the '-l' command line option). See also sections on MODULE and LOCALLABELCHAR directives.

Numeric Constants

Numeric constants must always begin with a decimal digit (thus hexadecimal constants that start with a letter must be prefixed by a '0' unless the '$' prefix is used). The radix is determined by a letter immediately following the digit string according to the following table:

Radix	Suffix	Prefix
2	B or b	%
8	O or o	@
10	D or d (or nothing)	
16	H or h	$

Decimal is the default radix, so decimal constants need no suffix or prefix.

The following representations are equivalent:

```
1234H          or     $1234
100d           or     100
177400o        or     @177400
01011000b      or     %01011000
```

The prefixes are provided for compatibility with some other source code formats but introduce a problem of ambiguity. Both '%' and '$' have alternate uses ('%' for modulo, '$' for location counter symbol). The ambiguity is resolved by examining the context. The '%' character is interpreted as the modulo operator only if it is in a position suitable for a binary operator. Similarly, if the first character following a '$' is a valid hexadecimal digit, it is assumed to be a radix specifier and not the location counter.

Character Constants

Character constants are single characters surrounded by single quotes. The ASCII value of the character in the quotes is returned. No escape provision exists to represent non-printable characters within the quotes, but this is not necessary since these can be just as easily represented as numeric constants (or using the TEXT directive which does allow escapes).

String Constants

String constants are one or more characters surrounded by double quotes. Note that string constants are not allowed in expressions. They are only allowable following the TITLE, BYTE, DB, and TEXT assembler directives. The quoted strings may also contain escape sequences to put in unprintable values. The following escape sequences are supported:

Escape Sequence	Description
\n	Line Feed
\r	Carriage return
\b	Backspace
\t	Tab
\f	Formfeed
\\	Backslash

\"	Quote
\000	Octal value of character

Location Counter Symbol

The current value of the location counter (PC) can be used in expressions by placing a '$' in the desired place. The Location Counter Symbol is allowable anywhere a numeric constant is. (Note that if the '$' is followed by a decimal digit then it is taken to be the hexadecimal radix indicator instead of the Location Counter symbol, as mentioned above). The '*' may also be used to represent the location counter, but is less preferred because of its ambiguity with the multiplicative operator.

Operators

Expressions can optionally contain operators to perform some alterations or calculations on particular values. The operators are summarized as follows:

Operator	Type	Description
+	**Additive**	addition
-		subtraction
*	**Multiplicative**	multiplication
/		division
%		modulo
<<		logical shift left
>>		logical shift right
~	**Unary**	bit inversion (one's complement)
-		unary negation
=	**Relational**	equal
==		equal
!=		not equal
<		less than
>		greater than
<=		less than or equal
>=		greater than or equal
&	**Binary**	binary 'and'
|		binary 'or'
^		binary 'exclusive or'

The syntax is much the same as in 'C' with the following notes.

- No operator precedence is in effect. Evaluation is from left to right unless grouped by parenthesis (see example below).
- All evaluations are done with 32 bit signed precision.
- Both '=' and '==' are allowable equality checkers. This is allowed since the syntax does not provide assignment capability (as '=' would normally imply).

The relational operators return a value of 1 if the relation is true and 0 if it is false. Thirty-two bit signed arithmetic is used.
It is always a good idea to explicitly indicate the desired order of evaluation with parenthesis, especially to maintain portability since TASM does not evaluate expressions in the same manner as many other assemblers. To understand how it does arrive at the values for expressions, consider the following example:

```
1 + 2*3 + 4
```

TASM would evaluate this as:

```
(((1 + 2) * 3) + 4) = 13
```

Typical rules of precedence would cause the (2*3) to be evaluated first, such as:

```
1 + (2*3) + 4       = 11
```

To make sure you get the desired order of evaluation, use parenthesis liberally. Here are some examples of valid expressions:

```
(0f800H + tab)
(label_2 >> 8)
(label_3 << 8) & $f000
$ + 4
010010000100100b + 'a'
(base + ((label_4 >> 5) & (mask << 2))
```


6.10. Assembler Directives

Most of the assembler directives have a format similar to the machine instruction format.

However, instead of specifying operations for the processor to carry out, the directives cause the assembler to perform some function related to the assembly process. TASM has two types of assembler directives - those that mimic the 'C' preprocessor functions, and those that resemble the more traditional assembler directive functions. Each of these will be discussed.

The 'C' preprocessor style directives are invoked with a '#' as the first character of the line followed by the appropriate directive (just as in 'C'). Thus, these directives cannot have a label preceding them (on the same line). Note that in the examples directives are shown in upper case, however, either upper or lower case is acceptable.

ADDINSTR

The ADDINSTR directive can be used to define additional instructions for TASM to use in this assembly. The format is:

```
[label] .ADDINSTR inst args opcode nbytes rule class shift binor
```

The fields are separated by white space just as they would appear in an instruction definition file. See the TASMTABS.TXT file on the TASM distribution disk for more detail.

AVSYM

See SYM/AVSYM.

BLOCK

The BLOCK directive causes the Instruction Pointer to advance the specified number of bytes without assigning values to the skipped over locations. The format is:

```
[label] .BLOCK        expr
```

Some valid examples are:

```
word1   .BLOCK    2
byte1   .block    1
buffer  .block    80
```

BSEG/CSEG/DSEG/NSEG/XSEG

These directives can be invoked to indicate the appropriate address space for symbols and labels defined in the subsequent code. The invocation of these directives in no way affects the code generated, only provides more information in the symbol table file if the AVSYM directive is employed. Segment control directives such as these are generally supported by assemblers that generate relocatable object code. TASM does not generate relocatable object code and does not support a link phase, so these directives have no direct effect on the resulting object code. The segments are defined as follows:

Directive	Segment Description
BSEG	Bit address
CSEG	Code address
DSEG	Data address (internal RAM)
NSEG	Number or constant (EQU)
XSEG	External data address (external RAM)

BYTE

The BYTE directive allows a value assignment to the byte pointed to by the current Instruction Pointer. The format is:

```
[label] .BYTE   expr [, expr ...]
```

Only the lower eight bits of *expr* are used. Multiple bytes may be assigned by separating them with commas or (for printable strings) enclosed in double quotes. Here are some examples:

```
label1   .BYTE     10010110B
         .byte     'a'
         .byte     0
         .byte     100010110b,'a',0
         .byte     "Hello", 10, 13, "World"
```

CHK

The CHK directive causes a checksum to be computed and deposited at the current location. The starting point of the checksum calculation is indicated as an argument. Here is the format:

```
[label]    .CHK    starting_addr
```

Here is an example:

```
start: NOP
       LDA #1
       .CHK start
```

The checksum is calculated as the simple arithmetic sum of all bytes starting at the *starting_addr* up to but not including the address of the CHK directive. The least significant byte is all that is used.

CODES/NOCODES

The CODES/NOCODES directives can be used to alternately turn on or off the generation of formatted listing output with line numbers, opcodes, data, etc. With NOCODES in effect, the source lines are sent to the listing file untouched. This is useful around blocks of comments that need a full 80 columns of width for clarity.

DB

This is alternate form of the BYTE directive.

DW

This is alternate form of the WORD directive.

DEFINE

The DEFINE directive is one of the most powerful of the directives and allows string substitution with optional arguments (macros). The format is as follows:

```
#DEFINE  macro_label[(arg_list)]  [macro_definition]
```

Where:

macro_label
character string to be expanded when found in the source file

arg_list
optional argument list for variable substitution in macro expansion

macro_def
string to replace the occurrences of *macro_label* in the source file.

The simplest form of the DEFINE directive might look like this:

```
#DEFINE MLABEL
```

Notice that no substitutionary string is specified. The purpose of a statement like this would typically be to define a label for the purpose of controlling some subsequent conditional assembly (IFDEF or IFNDEF).

A more complicated example, performing simple substitution, might look like this:

```
#DEFINE VAR1_LO(VAR1 & 255)
```

This statement would cause all occurrences of the string 'VAR1_LO' in the source to be substituted with '(VAR1 & 255)'.
As a more complicated example, using the argument expansion capability, consider this:

```
#DEFINE  ADD(xx,yy)    clc\ lda xx\ adc yy\ sta xx
```

If the source file then contained a line like this:

```
ADD(VARX,VARY)
```

It would be expanded to:

```
clc\ lda VARX\ adc VARY\ sta VARX
```

The above example shows the use of the backslash ('\') character as a multiple instruction statement delimiter. This approach allows the definition of fairly powerful, multiple statement macros. The example shown generates 6502 instructions to add one memory location to another.
Some rules associated with the argument list:

- Use a maximum of 10 arguments.
- Each argument should be a maximum of 15 characters.

Note that macros can be defined on the TASM command line, also, with the **-d** option flag.

DEFCONT

The DEFCONT directive can be used to add to the last macro started with a DEFINE directive. This provides a convenient way to define long macros without running off the

edge of the page. The ADD macro shown above could be defined as follows:

```
#DEFINE         ADD(xx,yy)      clc
#DEFCONT                        \ lda xx
#DEFCONT                        \ adc yy
#DEFCONT                        \ sta xx
```

ECHO

The ECHO directive can be used to send output to the console (stderr). It can accept either a quoted text string (with the standard escape sequences allowed) or a valid expression. It can accept only one or the other, however. Multiple instances of the directive may be used to create output that contains both. Consider the following example:

```
.ECHO "The size of the table is "
.ECHO (table_end - table_start)
.ECHO " bytes long.\n"
```

This would result in a single line of output something like this:

```
The size of the table is 196 bytes long.
```

EJECT

The EJECT directive can be used to force a top-of-form and the generation of a page header on the list file. It has no effect if the paging mode is off (see PAGE/NOPAGE). The format is:

```
    [label]     .EJECT
```

ELSE

The ELSE directive can optionally be used with IFDEF, IFNDEF and IF to delineate an alternate block of code to be assembled if the block immediately following the IFDEF, IFNDEF or IF is not assembled.

Here are some examples of the use of IFDEF, IFNDEF, IF, ELSE, and ENDIF:

```
#IFDEF   label1
    lda      byte1
    sta      byte2
#ENDIF

#ifdef   label1
    lda      byte1
#else
    lda      byte2
#endif

#ifndef  label1
    lda      byte2
#else
    lda      byte1
#endif

#if ($ >= 1000h)
; generate an invalid statement to cause an error
;  when we go over the 4K boundary.
!!! PROM bounds exceeded.
#endif
```

END

The END directive should follow all code/data generating statements in the source file. It forces the last record to be written to the object file. The format is:

```
    [label]       .END [addr]
```

The optional *addr* will appear in the last object record (Motorola S9 record type) if the object format is Motorola hex. The *addr* field is ignored for all other object formats.

ENDIF

The ENDIF directive must always follow an IFDEF, IFNDEF, or IF directive and signifies the end of the conditional block.

EQU

The EQU directive can be used to assign values to labels. The labels can then be used in expressions in place of the literal constant. The format is:

```
    label   .EQU   expr
```

Here is an example:

```
    MASK    .EQU  0F0H
    ;
```

```
        lda   IN_BYTE
        and   MASK
        sta   OUT_BYTE
```

An alternate form of the EQU directive is '='. The previous example is equivalent to any of the following:

```
MASK    =   0F0H
MASK    =0F0H
MASK    = $F0
```

White space must exist after the *label*, but none is required after the '='.

EXPORT

The EXPORT directive can be used to define labels (symbols) that are to be written to the export symbol file. The symbols are written as equates (using the .EQU directive) so that the resulting file can be included in a subsequent assembly. This feature can help overcome some of the deficiencies of TASM due to its lack of a relocating linker. The format is:

```
[label]  .EXPORT      label [,label...]
```

The following example illustrates the use of the EXPORT directive and the format of the resulting export file:

Source file:

```
EXPORT        read_byte
EXPORT        write_byte, open_file
```

Resulting export file:

```
read_byte      .EQU   $1243
write_byte     .EQU   $12AF
open_file      .EQU   $1301
```

FILL

The FILL directive can be used to fill a selected number of object bytes with a fixed value. Object memory is filled from the current program counter forward. The format is as follows:

```
[label]  .FILL      number_of_bytes [,fill_value]
```

The *number_of_bytes* value can be provided as any valid expression. The optional fill_value can also be any valid expression. If *fill_value* is not provided, a default value of 255 ($FF) is used.

IFDEF

The IFDEF directive can be used to optionally assemble a block of code. It has the following form:

```
#IFDEF  macro_label
```

When invoked, the list of macro labels (established via DEFINE directives) is searched. If the label is found, the following lines of code are assembled. If not found, the input file is skipped until an ENDIF or ELSE directive is found.
Lines that are skipped over still appear in the listing file, but a '~' will appear immediately after the current PC and no object code will be generated (this is applicable to IFDEF, IFNDEF, and IF).

IFNDEF

The IFNDEF directive is the opposite of the IFDEF directive. The block of code following is assembled only if the specified *macro_label* is undefined. It has the following form:

```
#IFNDEF  macro_label
```

When invoked, the list of macro labels (established via DEFINE directives) is searched. If the label is not found, the following lines of code are assembled. If it is found, the input file is skipped until an ENDIF or ELSE directive is found.

IF

The IF directive can be used to optionally assemble a block of code dependent on the value of a given expression. The format is as follows:

```
#IF     expr
```

If the expression *expr* evaluates to non-zero, the following block of code is assembled (until an ENDIF or ELSE is encountered).

INCLUDE

The INCLUDE directive reads in and assembles the indicated source file. INCLUDEs can be nested up to six levels. This allows a convenient means to keep common definitions, declarations, or subroutines in files to be included as needed. The format is as follows:

```
#INCLUDE        filename
```

The *filename* must be enclosed in double quotes. Here are some examples:

```
#INCLUDE "macros.h"
#include "equates"
#include "subs.asm"
```

LIST/NOLIST

The LIST and NOLIST directives can be used to alternately turn the output to the list file on (LIST) or off (NOLIST). The formats are:

```
.LIST
.NOLIST
```

LOCALLABELCHAR

The LOCALLABELCHAR directive can be used to override the default "_" as the label prefix indicating a local label. For example, to change the prefix to "?" do this:

```
[label]    .LOCALLABELCHAR "?"
```

Be careful to use only characters that are not operators for expression evaluation. To do so causes ambiguity for the expression evaluator. Some safe characters are "?", "{", and "}".

LSFIRST/MSFIRST

The LSFIRST and MSFIRST directives determine the byte order rule to be employed for the WORD directive. The default (whether correct or not) for all TASM versions is the least significant byte first (LSFIRST). The following illustrates its effect:

```
0000  34 12     .word $1234
0002            .msfirst
0002  12 34     .word $1234
0004            .lsfirst
0004  34 12     .word $1234
```

MODULE

The MODULE directive defines the scope of local labels. The format is:

```
[label]    .MODULE label
```

Here is an example:

```
       .MODULE module_x
       lda regx
       jne _skip
       dec
_skip  rts

       .MODULE module_y
       lda regy
       jne _skip
       dec
_skip  rts
```

In the above example, the local label *_skip* is reused without harm since the two usages are in separate modules. See also section LOCALLABELCHAR directive.

ORG

The ORG directive provides the means to set the Instruction Pointer (a.k.a. Program Counter) to the desired value. The format is:

```
[label] .ORG     expr
```

The *label* is optional. The Instruction pointer is assigned the value of the *expr*. For example, to generate code starting at address 1000H, the following could be done:

```
start   .ORG     1000H
```

The expression (*expr*) may contain references to the current Instruction Pointer, thus allowing various manipulations to be done. For example, to align the Instruction Pointer on the next 256 byte boundary, the following could be done:

```
.ORG   (($ + 0FFH) & 0FF00H)
```

ORG can also be used to reserve space without assigning values:

```
.ORG     $+8
```

An alternate form of ORG is '*=' or '$='. Thus the following two examples are exactly equivalent to the previous example:

```
*=*+8
$=$+8
```

PAGE/NOPAGE

The PAGE/NOPAGE directives can be used to alternately turn the paging mode on (PAGE) or off (NOPAGE). If paging is in effect, then every sixty lines of output will be followed by a Top of Form character and a two line header containing page number, filename, and the title. The format is:

```
.PAGE
.NOPAGE
```

The number of lines per page can be set with the '-p' command line option.

SET

The SET directive allows the value of an existing label to be changed. The format is:

```
label   .SET    expr
```

The use of the SET directive should be avoided since changing the value of a label can sometimes cause phase errors between pass 1 and pass 2 of the assembly.

SYM/AVSYM

These directives can be used to cause a symbol table file to be generated. The format is:

```
.SYM    ["symbol_filename"]
.AVSYM  ["symbol_filename"]
```

For example:

```
.SYM        "symbol.map"
.SYM
.AVSYM      "prog.sym"
.AVSYM
```

The two directives are similar, but result in a different format of the symbol table file. The format of the SYM file is one line per symbol, each symbol starts in the first column and is followed by white space and then four hexadecimal digits representing the value of the symbol. The following illustrates the format:

```
label1          FFFE
label2          FFFF
label3          1000
```

The AVSYM directive is provided to generate symbol tables compatible with the Avocet 8051 simulator. The format is similar, but each line is prefixed by an 'AS' and each symbol value is prefixed by a segment indicator:

```
AS      start           C:1000
AS      read_byte       C:1245
AS      write_byte      C:1280
AS      low_nib_mask    N:000F
AS      buffer          X:0080
```

The segment prefixes are determined by the most recent segment directive invoked (see BSEG/CSEG/DSEG/NSEG/XSEG directives).

TEXT

This directive allows an ASCII string to be used to assign values to a sequence of locations starting at the current Instruction Pointer. The format is:

```
[label] .TEXT   "string"
```

The ASCII value of each character in string is taken and assigned to the next sequential location. Some escape sequences are supported as follows:

Escape Sequence	Description
\n	Line Feed
\r	Carriage return
\b	Backspace
\t	Tab
\f	Formfeed
\\	Backslash
\"	Quote
\000	Octal value of character

Here are some examples:

```
message1    .TEXT   "Disk I/O error"
message2    .text   "Enter file name "
            .text   "abcdefg\n\r"
            .text   "I said \"NO\""
```

TITLE

The TITLE directive allows the user to define a title string that appears at the top of each

page of the list file (assuming the PAGE mode is on). The format is:

```
[label]    .TITLE  "string"
```

The *string* should not exceed 80 characters. Here are some examples:

```
.TITLE  "Controller version 1.1"
.title  "This is the title of the assembly"
.title  ""
```

WORD

The WORD directive allows a value assignment to the next two bytes pointed to by the current Instruction Pointer. The format is:

```
[label] .WORD expr [,expr...]
```

The least significant byte of expr is put at the current Instruction Pointer with the most significant byte at the next sequential location (unless the MSFIRST directive has been invoked). Here are some examples:

```
data_table      .WORD   (data_table + 1)
                .word   $1234
                .Word   (('x' - 'a')  << 2)
                .Word  12, 55, 32
```


6.11. Object file formats

Intel Hex Object Format

This is the default object file format. This format is line oriented and uses only printable ASCII characters except for the carriage return/line feed at the end of each line. The format is symbolically represented as:

```
:NN AAAA RR HH CC CRLF
```

Where:

:
Record Start Character (colon)

NN
Byte Count (2 hex digits)

AAAA
Address of first byte (4 hex digits)

RR
Record Type (**00** except for last record which is **01**)

HH
Data Bytes (a pair of hex digits for each byte of data in the record)

CC
Check Sum (2 hex digits)

CRLF
Line Terminator (CR/LF for DOS, LF for LINUX)

The last line of the file will be a record conforming to the above format with a byte count of zero.

The checksum is defined as:

```
sum = byte_count+address_hi+address_lo+record_type+(sum of all data bytes)
checksum = ((-sum) & ffh)
```

Here is a sample listing file followed by the resulting object file:

```
0001   0000
0002   1000                     .org   $1000
0003   1000 010203040506       .byte  1, 2, 3, 4, 5, 6, 7, 8
0003   1006 0708
0004   1008 090A0B0C0D0E       .byte  9,10,11,12,13,14,15,16
0004   100E 0F10
0005   1010 111213141516       .byte 17,18,19,20,21,22,23,24,25,26
0005   1016 1718191A
0006   101A                     .end
    :1810000001020304050607080 90A0B0C0D0E0F10111213141516 1718AC
    :02101800191AA3
    :00000001FF
```

Intel Hex Word Address Object Format

This format is identical to the **Intel Hex Object Format** except that the address for each line of object code is divided by two thus converting it to a word address (16 bit word). All other fields are identical.
Here is an example:

```
:1808000001020304050607080090A0B0C0D0E0F10111213141516171 8AC
:02080C00191AA3
:00000001FF
```

MOS Technology Hex Object Format

This format is line oriented and uses only printable ASCII characters except for the carriage return/line feed at the end of each line. Each line in the file is of the following format:
:NN AAAA HH CC CRLF

Where:

;
Record Start Character (semicolon)

NN
Byte Count (2 hex digits)

AAAA
Address of first byte (4 hex digits)

HH
Data Bytes (a pair of hex digits for each byte of data in the record)

CCCC
Check Sum (4 hex digits)

CRLF
Line Terminator (CR/LF for DOS, LF for LINUX)

The last line of the file will be a record conforming to the above format with a byte count of zero. The checksum is defined as:
sum =byte_count+address_hi+address_lo+record_type+(sum of all data bytes)
checksum = (sum & ffffh)

Here is a sample object file:

```
;1810000102030405060708090A0B0C0D0E0F10111213141516171801 54
;021018191A005D
;00
```

Motorola Hex Object Format

This format is line oriented and uses only printable ASCII characters except for the carriage return/line feed at the end of each line. The format is symbolically represented as:
S1 NN AAAA HH CCCC CRLF

Where:

S1
Record Start tag

NN
Byte Count (2 hex digits) (data byte count + 3)

AAAA
Address of first byte (4 hex digits)

HH
Data Bytes (a pair of hex digits for each byte of data in the record)

CC
Check Sum (2 hex digits)

CRLF
Line Terminator (CR/LF for DOS, LF for LINUX)

The checksum is defined as:
sum = byte_count+address_hi+address_lo+(sum of all data bytes)
checksum = ((~sum) & ffh)

Here is a sample file:

```
S11B10000102030405060708090A0B0C0D0E0F101112131415161718A8
S1051018191A9F
S9030000FC
```

The last line of the file will be a record with a byte count of zero and a tag of S9. The address field will be 0000 unless and address was provided with the END directive in which case it will appear in the address field.

Binary Object Format.

This file format is essentially a memory image of the object code without address, checksum or

format description information.
Note that when this object format is selected (-b option), the -c option is forced. This is done so that no ambiguity results from the lack of address information in the file. Without the -c option, discontinuous blocks of object code would appear contiguous.

6.12. Listing file format

Each line of source code generates one (or more) lines of output in the listing file. The fields of the output line are as follows:

- Current source file line number (4 decimal digits).
- An optional '+' appears if this is an 'INCLUDE' file. (One '+' for each level of INCLUDE invoked).
- Current Instruction Pointer (4 hex digits). An optional '~' follows the Instruction Pointer if the line of source code is not being assembled because of an IFDEF, IFNDEF, or IF directive.
- Resulting code/data generated from this source line (two hex digits per byte, each byte separated by a space, up to six bytes per line).
- The source line exactly as it appears in the source file.

If paging is enabled (by either the '-p' option flag or the .PAGE directive) some additional fields will be inserted into the listing file every 60 lines. These fields are:

- Top of Form (form feed).
- Assembler identifier (e.g. "TASM 6502 Assembler").
- Initial source file name.
- Page number.
- Title.

If errors are encountered, then error messages will be interspersed in the listing. TASM outputs error messages proceeding the offending line. The following example illustrates the format:

```
0001   0000             label1  .equ  40h
0002   0000             label2  .equ  44h
0003   0000
0004   1000             start:  .org  1000h
0005   1000 E6 40               inc   label1
0006   1002 E6 44               inc   label2
tt.asm line 0007: Label not found: (label3)
0007   1004 EE 00 00            inc   label3
0008   1007 4C 00 10            jmp   start
0009   100A                     .end
0010   100A
tasm: Number of errors = 1
```

TASM. Copyright (C) 1998 by Squak Valley Software.
All rights reserved.

6.13. PROM Programming

A wide variety of PROM programming equipment is available that can use object code in one or more of the formats supported by TASM. Here are some notes concerning the generation of code to be programmed into PROMs:

PRESET MEMORY

It is often desirable to have all bytes in the PROM programmed even if not explicitly assigned a value in the source code (e.g. the bytes are skipped over with a .ORG statement). This can be accomplished by using the *-c* (contiguous block) and the *-f* (fill) command line option flags. The *-c* will ensure that every byte from the lowest byte assigned a value to the highest byte assigned a value will be in the object file with no gaps. The *-f* flag will assign the specified value to all bytes before the assembly begins so that when the object file is written, all bytes not assigned a value in the source code will have a known value. As an example, the following command line will generate object code in the default Intel Hex format with all bytes not assigned a value in the source set to EA (hex, 6502 NOP instruction):

```
tasm -65 -c -fEA test.asm
```

CONTIGUOUS BLOCKS

To ensure that TASM generates object code to cover the full address range of the target PROM, put a .ORG statement at the end of the source file set to the last address desired. For example, to generate code to be put in a 2716 EPROM (2 Kbytes) from hex address $1000 to $17ff, do something like this in the source file:

```
;start of the file
    .ORG    $1000
```

- TeddyWareZ -

```
;rest of the source code follows

[source code]

;end of the source code
    .ORG    $17ff
    .BYTE   0
    .END
```

Now, to invoke TASM to generate the code in the binary format with all unassigned bytes set to 00 (6502 BRK instruction), do the following:

```
tasm -65 -b -f00 test.asm
```

(Note that -b forces the -c option.)

6.14. Error messages

Error Message Format

TASM error messages take the following general form:

```
filename line line_number: error_message
```

For example:

```
main.asm line 0032: Duplicate label (start)
```

This format is compatible with the Brief editor (from Borland International). Brief provides the ability to run assemblies from within the editor. Upon completion of the assembly, Brief will parse the error messages and jump to each offending line in the source file allowing the user to make corrective edits.

To use this feature, it is necessary to configure a Brief environment variable to specify the assembly command associated with source files that end in .asm:

```
SET BCASM = "tasm %s.asm"
```

TASM also needs to know the proper table to use. It can be added above, or the TASMOPTS environment variable can be used separately:

```
SET TASMOPTS=-65
```

ERROR MESSAGE DESCRIPTIONS

Binary operator where value expected
Two binary operators in a row indicate a missing value.

Branch off of current 2K page
An instruction is attempting to branch to a location not within the current 2K byte page.

Branch off of current page
An instruction is attempting to branch to a location not within the current 256 byte page.

Cannot malloc for label storage
Insufficient memory to store more labels. See **LIMITATIONS**.

Duplicate label
The label for the current line has already been assigned a value. Duplicate label checks are optionally enabled by the *-a* option.

File name too short
A file name on the command line is fewer than 3 characters. A two character file name may be valid, of course, but it is detected as an error to prevent a garbled option flag from being taken as a source file, which in turn can result in the true source file being taken as the object file. Since the object file is truncated at startup time, the source file could be clobbered.

Forward reference in equate
An EQU directive is using a label on the right hand side that has not yet been defined.

Heap overflow on label definition
TASM was unable to allocate memory to store the label.

Imbalanced conditional.
An end-of-file was encountered at which time the level of descent in conditional directives was different from when the file was entered. Conditional directives include IF, IFDEF, and IFNDEF.

Invalid Object file type.
An object file type was requested by the -g command line option that is not valid. See section on Option g - Object File Format.

Invalid operand.
No indirection for this instruction. The first character of an operand was a left parenthesis for an instruction that does not explicitly specify that as the format. Some micros use the parenthesis as an indicator of indirection, but putting a layer of parenthesis around an expression is always a valid thing to do (as far as the expression evaluator is concerned). The test for this case is only done if the *-a4* option is selected. See section on **ASSEMBLY CONTROL**.

Invalid token where value expected.
Two binary operators in a row are not allowed.

Label too long.
Labels are limited to 31 characters.

Label value misaligned
The value of a label appears to have a different value on the second pass then it was computed to have on the first pass. This is generally due to Zero Page Addressing mode problems with the 6502 version of TASM. Labels that are used in operands for statements that could utilize Zero Page addressing mode should always be defined before used as an operand.

Label not found
A label used in an expression was not found in the current label table.

Label must pre-exist for SET.
The SET directive can only be applied to an existing label.

Label table overflow
To many labels have been encountered.

List file open error
TASM was not able to open the specified list file.

Macro expansion too long.
The expansion of a macro resulted in a line that exceeded the maximum length.

Max number of nested conditionals exceeded
Too many levels of IF, IFDEF, or IFNDEF.

Maximum number of args exceeded
Too many macro arguments.

Maximum number of macros exceeded
Too many macros (DEFINEs) have been encountered.

No END directive before EOF
The source file did not have an END directive in it. This is not fatal, but may cause the last object file record to be lost.

No files specified
TASM was invoked with no source file specified.

No such label
A SET directive was encountered for a label not yet defined. The value of labels that are modified by the SET directive must already exist.

No terminating quote
A double quote was used at the start of a a text string but was not used at the end of the string.

No indirection for this instruction.
A parenthesis was found around the operand expression. This may indicate an attempt to use indirection where it is inappropriate.

Non-unary operator at start of expression
A binary operator (such as '*') was found at the beginning of an expression. Some micros use '*' as an indirection operator. Since it is also a legitimate operator in an expression, some ambiguity can arise. If a particular instruction/addressing mode does not allow indirection, and a '*' is placed in front of the associated expression, the assembler will assume this error. See the *-a8* option of ASSEMBLY CONTROL.

Object file open error
TASM was not able to open the specified object file.

Range of argument exceeded
The value of an argument exceeds the valid range for the current instruction and addressing mode.

Range of relative branch exceeded
A branch instruction exceeds the maximum range.

Source file open error
TASM was not able to open the specified source file.

Unrecognized directive
A statement starting with a '.' or '#' has a mnemonic that is not defined as a directive.

Unrecognized instruction
A statement has an opcode mnemonic that is not defined.

Unrecognized argument
A statement has an operand format that is not defined.

Unknown token
Unexpected characters were encountered while parsing an expression.

Unused data in MS byte of argument
An instruction or directive used the least significant byte of an argument and left the most significant byte unused, but it was non-zero.

Unknown option Flag.
Invalid option flag has been specified on the command line. invoke TASM with nothing on the command line to see a list of valid options.

6.15. Limitations

Maximum number of labels	15000
Maximum length of labels	32 characters
Maximum address space	64 Kbytes (65536 bytes)
Maximum number of nested INCLUDES	4
Maximum length of TITLE string	79 characters
Maximum source line length	255 characters
Maximum length after macro expansion	255 characters
Maximum length of expressions	255 characters
Maximum length of pathnames	79 characters
Maximum length of command line	127 characters
Maximum number of instructions (per table)	1200
Maximum number of macros	1000
Maximum number of macro arguments	10
Maximum length of macro argument	16 characters
Memory requirements	512K

Other Limitations

- The 8048 version of TASM does not check for use of memory beyond any reasonable bounds (e.g. an 8048 has a maximum address space of 4 Kbytes but TASM will let you pretend that you have 64 Kbytes).
- Expression evaluation has no operator precedence in effect which can make for unexpected results if not explicitly grouped with parenthesis.
- First page of listing file will not show a user defined title (defined via TITLE directive).

TASM. Copyright (C) 1998 by Squak Valley Software.
All rights reserved.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z