1. General: <Up>

This assembler debugger is based on the 8051-family. Every command line can be iterated manually or automaticly (See part 'Simulator').The simulator ascts as a real processor.

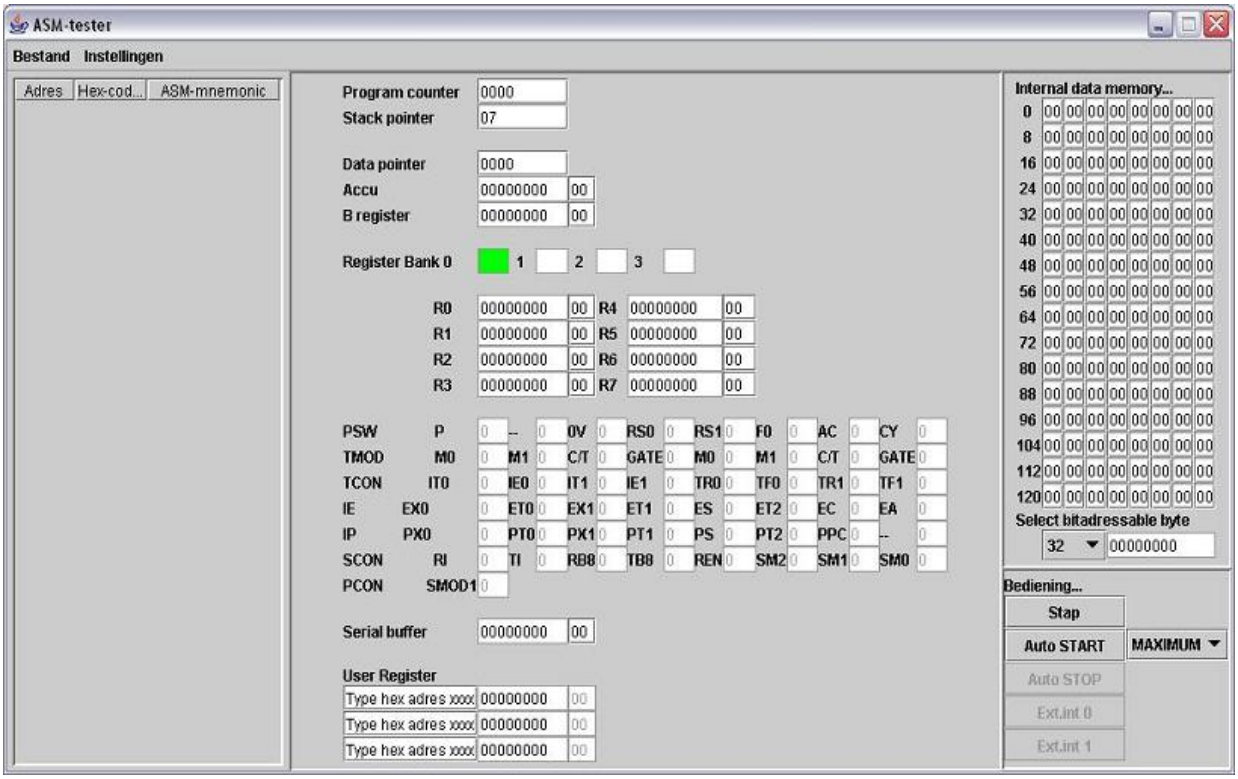
Practical is it possible to change about any visible value , the processor will adept to this new value (See part 'Changing values')

The programs has the possibility to simulate timers and interrupts (See part 'Timers and Interrupts')as well as interrupt priority's.

Full logging to text file of the tested programm inclusief time line in µsec in function of choosen cristzall. (See part 'Logging').

Screenshots of this english manual are in dutch, the program self is changed so that every text is in english

2. Opening screen (zie fig.1): <Up>



Figuur 1: Opening screen

There are the followin important parts

- Above the menu
- Left on the screen will be the list of the programm to be tested (empty on startup) it will be build out of the following colums : Adress - Hex Code - ASM Mnemonic.

- In the middel of the screen there are 'Special Function Register' the most important one's.
- Just At the top right we see the internal first 128 bytes
- Just below the internal bytes we can select de adresses ho are bit adressable, we get to see the binary value
- At the right bottom there's the control panel

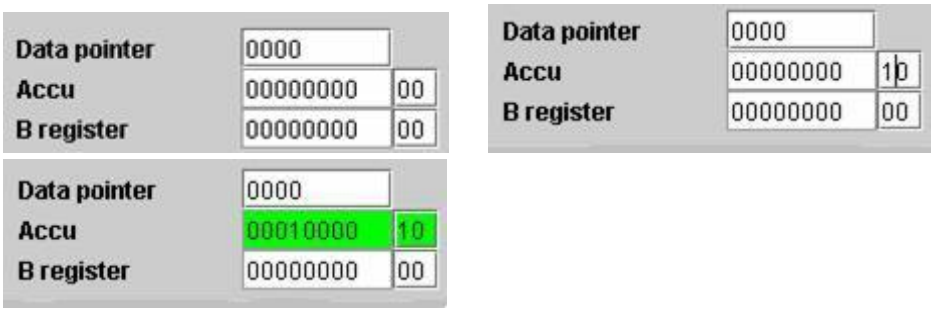
3. Changing values : <Up>

The user has the posibility to change the valeus shown in the 'SFR'-panel en the 'Internal Registers' – panel. It is only possible to change de hexadcimal values, de binary ones will be updated automatically.

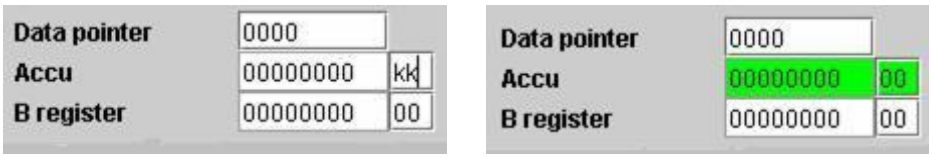
Teh entered value has to be acknowledged by 'Enter' or it will not be used by the program.

The entered value will be validated , when the value entered is not in bounds of possible values , the user will be warned en automatically will the value be replaced with the one that was present before the user changed.

A correct entered value will be accepted and the background colors green, also are the binary values adjusted to the new value. In the figures 2 you can see the mentioned above when a correct value is entered, the figures 3 shows what happens when a wrong value is entered.

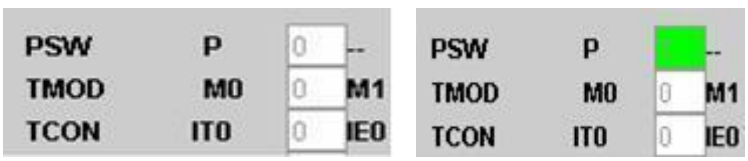


Figuur 2: Entering a good value



Figuur 3: I don't think kk is a hexadceimal value

the bit fields in the SFR panel are easily changed by clicking on them with the mouse. See figure 4



Figuur 4: Clicking on one of the bitfields

The part 'User Register'(see figure 5), When the user uses a datapointer (to adress an LCD or etc)it can be useful to see what value is actually send to that adress,ore on the other hand to simlulate some digital inputs, well we have to possibility to add three of these adresses.

-It is not possible to enter an adress twice

-The data on an adress can only be changed after a correct one is entered.

-Also this time you only change the hex value de binary one is changed automatically.

Figuur 5: User registers

4. Simulator: <Up>

4.1 Openning a test program

Our debugger accepts as input 'intel-hex files', our debugger will make the translations needed. In order to do this he will be needing the three *.cvs files which contain the data. When one of this files is missing the simulator will show a message and the translation will be arborted.

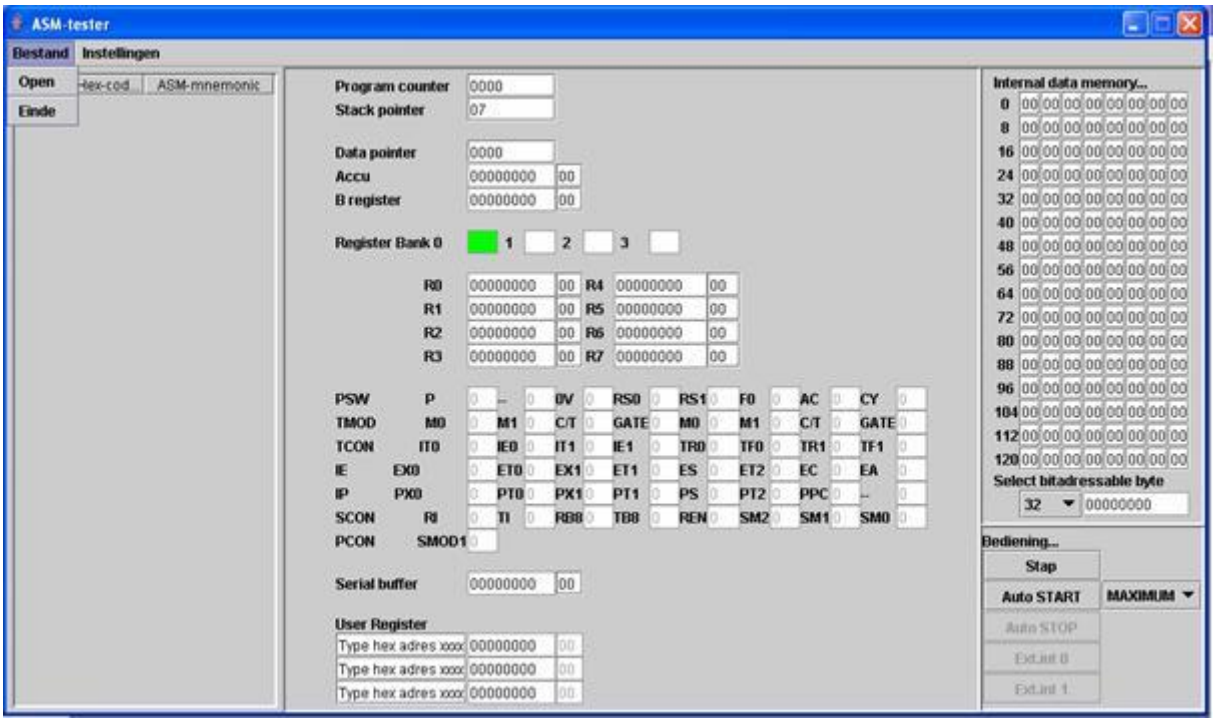
The three files needed are:
- -databestand.csv
- -sfr.csv
- -bit.csv

The user has the possibility to adjust these files , i suggest to make a copy for you do so, the files are of type comma seperated value, so they can be watched verry easily with programms such as 'Ecell'.

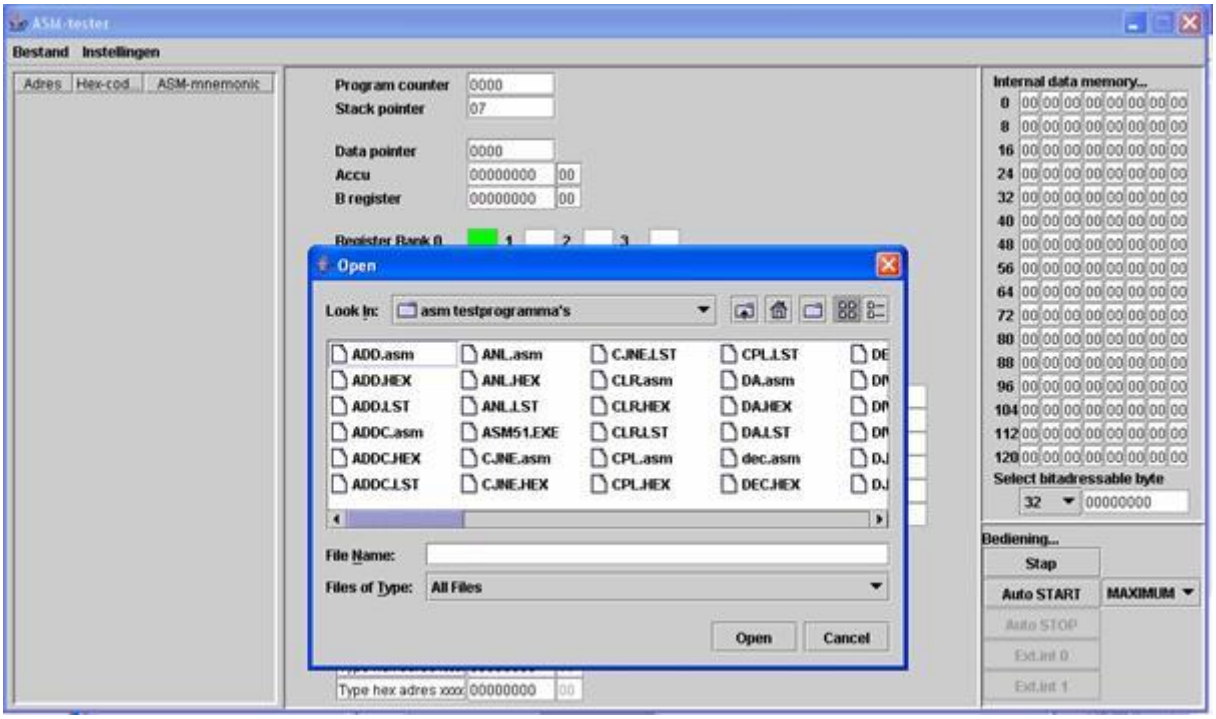
While the simulator is translating , it will also verify the hex file on errors, if one of these errors is found , a message box will appear on screen and the translation will be aborted.

If the translation was succesful , it will be showed on screen, immediatly the user will be asked for a start adress. You have to fill inan adress to continue and it has to be a valid one (which means it has to be in the left column of the screen).

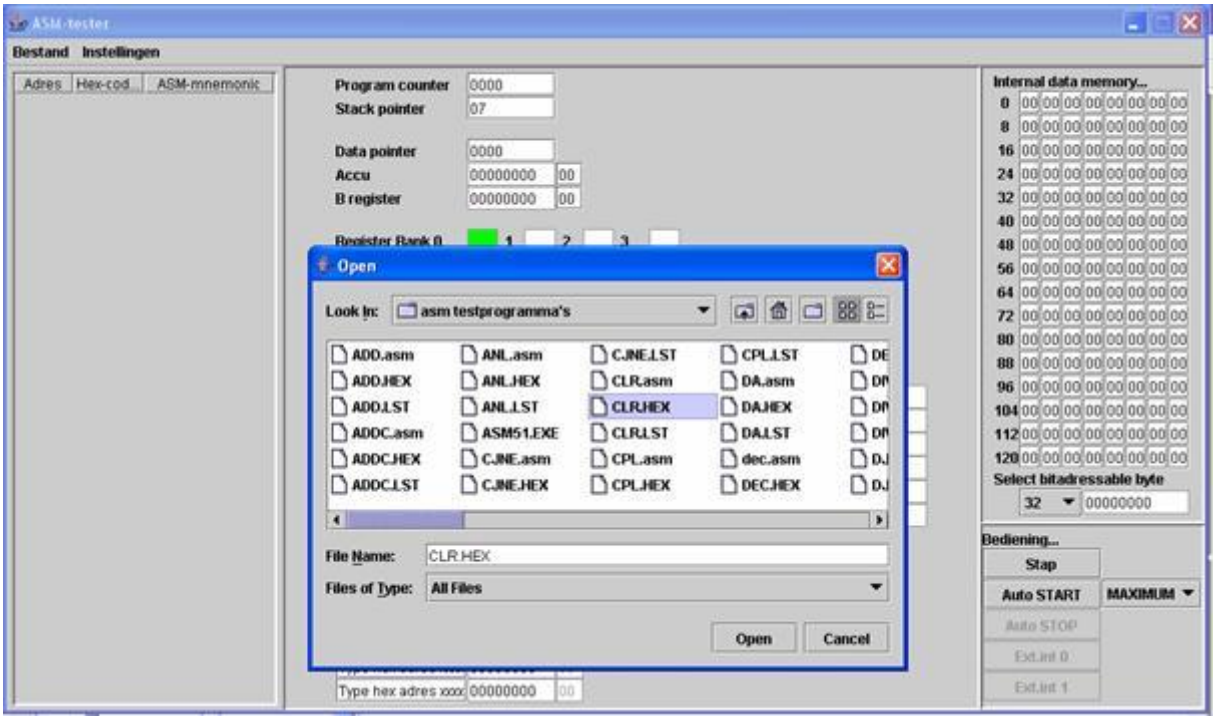
Teh filename of the opend file is on top of the program just above the menu.



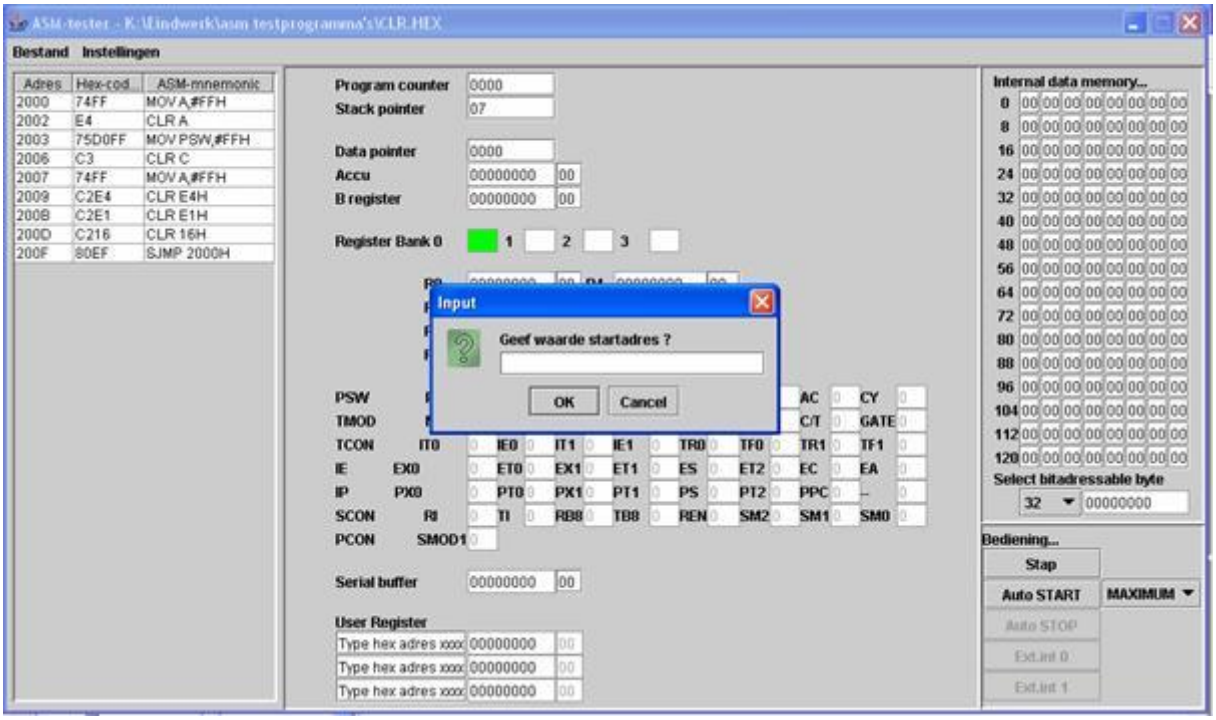
Figuur 6: Choose File - Open



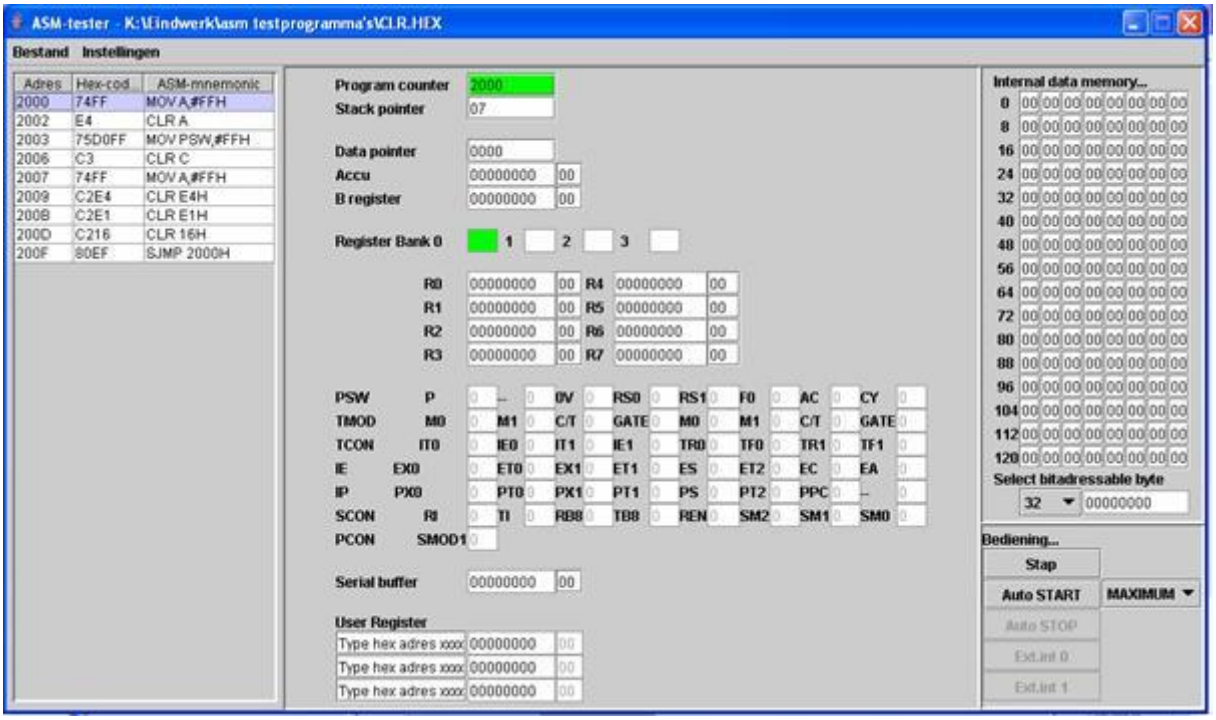
Figuur 7: Select the correct directory and choose a *.hex file



Figuur 8 :Select the .hex file



Figuur 9: Give an apropriate start adress



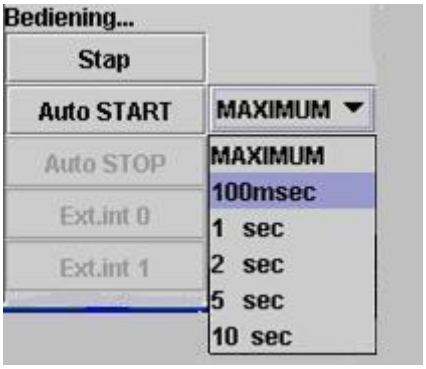
Figuur 10: The simulator is ready to do some testing

4.2 Testing

There are to posibilities to test our program , we have the manual mode or the auto mode.
In both of them you can change almost any register on the screen, we have to be carefull when we do this in auto-mode, it is possible to change the timer mode of a timer , but these effects will only take place after the timer is stopped and restarted !

In ‘Manuaee’- mode the timer and interrupt functions are disabled.(which means there will be now interrupts generated , and the timers doesn count , you have yo manually trigger an overflow)

In auto mode we have to select a speed of iteration over the program. Be careful this is the time the simulator will wait before executing the next command (zie figure 11).



Figuur 11: The speed selection

The simulator does not have the posibilities to perform test on actual speeds ike a controller does, when we choose the maximum speed we say to the controller there will be no wit time before executing the following command, however the speed that will be between to

commands is depended on te machine you are running on.

For example on a PC Pentium 4 speed 2,66GHz RAM memory:256MB we become approximately a maximum speed variating between 15 and 25msec per command.

In order to know if a timer does overflow after the time intended, or when we have to jnow how long an interrupt takes, we have to look at the log file (See part 'Logging') In order to have a correct timing diagram we have to select the cristall we are using, we can find this under settings in the menu.

Also the number of machinecycles has to be changed to the right value,see figure 12

Noot We do know out of theory dat a command takes one ore more machinecycli, most of the time a machinecycle takes 12 clock pulses from the cristall, some of the new ic's have a lower amount of clock pulses in order to achieve a better speed.

Figuur 12: Settings for cristall and machinecycli.

The number of machinecycli is easy to change , click on it and the new value will bea asked, you have to do this every time the programm is restarted.

The other controls (on the right bottom of the screen) are verry easy the button 'Step' will iterate over the programm every time you click on it.

The button 'AUTO Start' will iterate automatically according to the choosen speed, so choose speed first before pressing, there is also a button 'AUTO Stop' i don't have to tell you what's it's doing.

There are also 2 buttons for external interrupt clicking one of the mwill generate an interrupt , if the according vector adress is not programmed, there will appaer a message, and the programm wil continue , the button is deactivated for the time the interrupt is active.

When we reach the end of our programm a message is shown to the user, if the message appears before the end of the programm it means there was a jump to an adress not known to the processor.

5. Timers en interrupts: <Up>

A few important notes concerning timers for our simulator:

A timer works on the following principle, our controller has only one process running, wich means that our timer (which is a 'n up-counter) will add 1 every machine cycle.

The maximum value that a timer can have is FFFFh of 65535 decimal. (When we consider a timer in Mode 2)

With a controller with a cristall 12MHz and a cycle of 12 clockpulses, we come to the following equation:

1 / 12 000 000 * 12 klokpulsen = 1µsec
This means an overflow every 0.065sec.

If we wont to make a timer of 1 sec. we have to detect this overflow about 15 times.

In our simulator we have other speeds to recon with , if we suppose the maximum speed is about 15msec as mentioned before we become the following equation:

0.015 * 65535 = overflow will appear every 935sec

Not only will we have to wiat this long for an overflow, our logging will be huge.

Well interrupts is a whole different story, i can not spend this manual on how interrupts work, or what they do, there are many other sites ho can tell you, 8052 is one of them , it can be found on the section links. Anything that apply's to interrupt cna be simulated with our simulator

6. Logging <Up>

As mentioned before the tested program is logged to a text file. When the program was made there was not much time to develop a better way of logging to file, in the future this will be perhaps one of the tings that will be revised first.

In the example in figure 13 there is a preview of what to expect from this logging Besides normal program flow all the error message will also be logged so the user can evaluate what happend or what went wrong.

```
2004-05-10 18:12:53,187 DEBUG HoofdProgramma - Start HoofdProgramma
2004-05-10 18:12:53,187 DEBUG HoofdProgramma - modellen worden aangemaakt
2004-05-10 18:12:53,203 DEBUG HoofdProgramma - GUI wordt gebouwd
2004-05-10 18:13:05,750 DEBUG HoofdProgramma - inlezen file  -> K:\Eindwerk\asm
testprogramma's\ADD.HEX
2004-05-10 18:13:05,750 DEBUG HoofdProgramma - vertaal file  -> K:\Eindwerk\asm
testprogramma's\ADD.HEX
.          : MOV  Commando - Type 1 - MOV A,#10
.          : ADD  Commando - Type 1 - ADD 10,#30
.          : MOV  Commando - Type 1 - MOV A,#40
.          : MOV  Commando - Type 6 - MOV  30,#40
.          : ADD  Commando - Type 2 - ADD 40,40
.          : MOV  Commando - Type 1 - MOV A,#FF
.          : MOV  Commando - Type 6 - MOV  30,#01
.          : MOV  Commando - Type 14- MOV 0,#48
.          : ADD  Commando - Type 3 - ADD FF,01
.          : MOV  Commando - Type 1 - MOV A,#78
```

```
.          : MOV  Commando - Type 14- MOV 2,#4F
.          : ADD  Commando - Type 4 - ADD 78,4F
.          : MOV  Commando - Type 1 - MOV A,#53
.          : MOV  Commando - Type 14- MOV 4,#F0
.          : ADD  Commando - Type 4 - ADD 53,F0
.          : SJMP Commando - Type x - SJMP 100

2004-05-10 18:13:17,906 DEBUG HoofdProgramma - Start HoofdProgramma
2004-05-10 18:13:17,906 DEBUG HoofdProgramma - modellen worden aangemaakt
2004-05-10 18:13:17,921 DEBUG HoofdProgramma - GUI wordt gebouwd
2004-05-10 18:13:34,843 DEBUG HoofdProgramma - inlezen file  -> K:\Eindwerk\asm
testprogramma's\MOVX.HEX
2004-05-10 18:13:34,843 DEBUG HoofdProgramma - vertaal file  -> K:\Eindwerk\asm
testprogramma's\MOVX.HEX
Auto - Cylus : 2 CPU-tijd : 2.170021157706288
.          : MOV  Commando - Type 16- MOV  DPTR,#6000
Auto - Cylus : 4 CPU-tijd : 4.340042315412576
.          : MOV  Commando - Type 6 - MOV  30,#04
Auto - Cylus : 6 CPU-tijd : 6.510063473118864
.          : MOV  Commando - Type 6 - MOV  31,#05
Auto - Cylus : 7 CPU-tijd : 7.5950740519720075
.          : MOV  Commando - Type 14- MOV 0,#48
Auto - Cylus : 8 CPU-tijd : 8.680084630825151
.          : MOV  Commando - Type 14- MOV 1,#49
Auto - Cylus : 10 CPU-tijd : 10.850105788531438
.          : MOVX Commando - Type 0 - MOVX A,04
Auto - Cylus : 12 CPU-tijd : 13.020126946237726
.          : MOVX Commando - Type 0 - MOVX A,05
Auto - Cylus : 14 CPU-tijd : 15.190148103944013
.          : MOVX Commando - Type 1 - MOVX A,@DPTR
Auto - Cylus : 16 CPU-tijd : 17.360169261650302
.          : MOVX Commando - Type 2 - MOVX @Ri,A
Auto - Cylus : 18 CPU-tijd : 19.53019041935659
.          : MOVX Commando - Type 2 - MOVX @Ri,A
Auto - Cylus : 19 CPU-tijd : 20.615200998209733
.          : MOV  Commando - Type 1 - MOV A,#FF
Auto - Cylus : 21 CPU-tijd : 22.78522215591602
.          : MOVX Commando - Type 3 - MOVX @DPTR,A
```

Figuur 13 :The Logging file