THE Z80 BACK END TABLE

Frans van Haarlem

1. INTRODUCTION

This table was written to make it run, not to make it clever! The effect is, that the table written for the intel 8080, which was made very clever runs faster and requiers less space!! So, for anyone to run programs on a z80 machine: n attempt could be made to make this table as clever as the one for the i80, or the i80 table could be used, for that can run on every z80 too.

2. IMPLEMENTATION

It will not be possible to run the entire Amsterdam Compiler Kit on a Z80-based computer system. One has to write a program on another system, a system where the compiler kit runs on. This program may be a mixture of high-level languages, such as C or Pascal, EM and z80 assembly code. The program should be compiled using the compiler kit, producing z80 machine code. This code should come available to the z80 machine for example by downloading or by storing it in ROM (Read Only Memory). Depending on the characteristics of the particular z80 based system, some adaptions have to be made:

1) In *head_em*: the base address, which is the address where the first z80 instruction will be stored, and the initial value of the stackpointer are set to 0x1000 and 0x7ffe respectivally. The latter because it could run on a 32K machine as well. Other systems require other values.

2) In *head_em*: before calling "__m_a_i_n", the environment pointer, argument vector and argument count will have to be pushed onto the stack. Since this back-end is tested on a system without any knowledge of these things, dummies are pushed now.

3) In *tail_em*: proper routines "putchar" and "getchar" should be provided. They should write resp. read a character on/from the monitor. Maybe some conversions will have to be made. The ones for the Nascom and Hermac z80 micro's are to be found in the EM-library.

4) In *head_em*: an application program returns control to the monitor by jumping to address 0x20. Thie may have to be changed on different systems. For an CPM-machine for example this should be 0x5, to provide a warm boot.

5) In *tail_em*: the current version of the z80 back-end has very limited I/O capabilities, because it was tested on a system that had no knowlegde of files. So the implementation of the EM-instruction *mon* is very simple; it can only do the following things:

Monitor call 1:
    Exit
Monitor call 3:
    read, always reads from the monitor.
    echos the read character.
    ignores file descriptor.
Monitor call 4:
    write, always writes on the monitor.
    ignores file descriptor.
Monitor call 5:
    open file, returns file descriptor -1.
Monitor call 6:
    close file, returns error code = 0.
Monitor call 54:
    io-control, returns error code = 0.

If the system should do file-handling the routine ".mon" should be extended thoroughly.