Version 2, September 2009. **CCS-M1X3**

EMIDEC 1100 instruction set and instruction times.

General.
The EMIDEC 1100 has a word length of 36 bit, uses two-complement arithmetic and employs 6-bit characters. The primary memory *(immediate-access store)* consists of 1Kwords of magnetic core storage, giving random-access capability. An upgrade to 4K words of primary memory is available for the EMIDEC 1101, though addressing then becomes conceptually equivalent to four-word blocks. (In detail, the c1 field in an instruction (see below) is called into play to extend the addressing range).

The EMIDEC 1100 uses a *two-address* instruction format, with one instruction per word. The layout is as follows:

Function	a-address	b-address	c1	c2	mod
5	10	10	4	4	3

For many instructions, the a-address is a source and the b-address is a destination. ‘mod’ indicates one of seven modifier registers. These modifiers are amongst the EMIDEC 1100’s group of seventeen special registers, allocated as follows:

Special registers

0	permanent value of 0
1-7	modifiers
8-9	used by multiply/divide instructions (contains the double-length answer)
10	result of a collate (ie AND) instruction
11	permanent value of 1
12	permanent value of 1 in the a-address field – (ie bit 2^8 is set)
13	permanent value of 1 in the b-address field – (ie bit 2^{18} is set)
14	permanent value of 1 in the c-address field – (ie bit 2^{28} is set)
15	permanent value of 16 in the c-address – (ie bit 2^{35} is set)
16	console register - can have its value set by the program or set manually by the switches on the operator’s console.

The complete listing of EMIDEC 1100 instructions follows on the next page. This has been taken from: http://www.emidec.org.uk/ Typical times are given for sample instructions.

0 Conditional halt - the program will halt if the binary value of the a-address is non-zero and matches the console switch settings for "conditional halt"; a value of 1 in the c-address causes the alarm to sound; a value of 2 in c-address switches off the alarm.

1 Move the contents of a-address to b-address. ***Time: 120 microseconds.***

2 Shift left the contents of a-address by the number of places specified in c1-address, and place the result in b-address.

3 Shift right the contents of a-address by the number of places specified in c1-address, and place the result in b-address.

4 Double-length shift left - shift left the contents of a-address and the register next to it (as a double length register) by the number of places specified in c1-address, and place the result in b-address and the register next to it.

5 Double-length shift right - shift right the contents of a-address and the register next to it (as a double length register) by the number of places specified in c1-address, and place the result in b-address and the register next to it.

6 Collate - compare the a-address register with the b-address register, and place into special register 10 a copy of those bits which are on in both. Useful for extracting a portion of a word. ***Time: 140 microseconds.***

7 Add the contents of a-address to the contents of b-address. ***Time: 140 microseconds.***

8 Subtract the contents of a-address from the contents of b-address. ***Time: 150 microseconds.***

9 Multiply the contents of a-address by the contents of b-address, placing the result in registers 8 and 9 (as a double-register value). ***Time: 1260 microseconds.***

10 Divide the contents of registers 8 and 9 (as a double-register value) by the contents of a-address and place the result in b-address, and the remainder in register 9. ***Time: 1440 microseconds.***

11 Test zero - If a-address is zero, jump to the instruction indicated by b-address. The instruction 11 0 Rxxx was typically used as an unconditional jump or "Go to".

12 Test non-zero - If a-address is non-zero, jump to the instruction indicated by b-address

13 Test positive - If a-address is positive (ie digit 35 is zero), jump to the instruction indicated by b-address ***Time: 150 microseconds.***

14 Test negative - If a-address is negative (ie digit 35 is "1"), jump to the instruction indicated by b-address

15 Transfer from drum - transfer c2-address 4-word blocks from the drum address specified in b+c1-address to core store starting at a-address. A specific use of this instruction is 15 S 3 ... load into storage the program stage indicated by b-address. ***(See note below for times)***

16 Transfer to drum - transfer c2-address 4-word blocks from core store starting at a-address to the drum address specified in b+c1-address ***(See note below for times)***

17 Input block transfer - transfer the contents of the buffer unit of the input channel specified in c2-address to 16 consecutive registers starting at a-address. ***Time: 1190 microseconds.***

18 Output block transfer - transfer to the buffer unit of the output channel specified in c2-address the 16 consecutive registers starting at a-address. ***Time: 1130 microseconds.***

19 Convert decimal input - take c1-address characters from the buffer unit of the input channel specified in c2-address, and place the result as a binary number in the register specified by a-address.

20 Convert sterling input - take c1-address characters from the buffer unit of the input channel specified in c2-address, convert the data from sterling (£sd) to decimal pence and place the result as a binary number in the register specified by a-address.

21 Alphanumeric input - take c1-address characters (max 12) from the buffer unit of the input channel specified in c2-address, and place the result as 6-bit coded characters in the register specified by a-address and the one next to it. Any unused character positions are filled with 'no character' code (100000). ***Time: 340 microsecs.***

22 Convert decimal output - take the contents of a-address, convert them to a decimal number, and transfer c1-address characters to the buffer unit of the output channel specified in c2-address. ***Time: 2990 microsecs.***

23 Convert sterling output - take the contents of a-address, convert them from binary pence to sterling, and transfer c1-address characters to the buffer unit of the output channel specified in c2-address. **3220 microsecs.**

24 Alphanumeric output - take c1-address characters (max 12) from the register specified by a-address and the one next to it to the buffer unit of the output channel specified in c2-address. ***Time: 290 microseconds.***

25 Count test - subtract the contents of a-address register from register 7, and if the value of register 7 is non-zero, jump to the instruction indicated by b-address.

26 Double-length add - add the contents of a-address and the next register (as a double-length register) to the contents of b-address and the next register. ***Time: 290 microseconds.***

27 Double-length subtract - subtract the contents of a-address and the next register (as a double-length register) from the contents of b-address and the next register. ***Time: 310 microseconds.***

28 Sterling output with spaces - output the value specified in a-address, converted from decimal pence to pounds shillings and pence, and inserting one space between each column. Thus, a value of 258 would be output as 1 1 6 (one pound, one shilling and sixpence). Since the maximum number of output characters is restricted to 12, the maximum output value is 9999999 19 11; for larger numbers, the most significant digits will be lost. Specify the required number of output characters (including spaces) in the c1-address, and the required output channel in the c2-address. ***Time: 3720 microseconds.***

29 Set link and jump - places the contents of its own a-address into the b-address of register 7 and jumps to the instruction specified in its own b-address; typically this was used to perform a subroutine and return to the address specified in its own a-address (the subroutine would end with an unconditional jump to 0 but modified by modifier - ie register - 7). ***Time: 110 microseconds.***

30 Negative sum - places the complement of c1-address registers, starting at a-address, into b-address. Typically, this instruction was used in parity-checking, especially for mag tape, but could also be used, cutely, for totalling a set of numbers.

31 Unconditional halt - if bit-n is set, the alarm will sound when the program halts. Normally used to indicate that the program has finished, with all fields set to their highest values, so that all lights show in the last-instruction-register on the console.

As well as the basic instruction set, there are a number of "dummy" instructions... see examples below.

- **90 Set up an alphanumeric constant** - an instruction (ie one 36-bit word) can contain up to 6 characters (each using 6 bits); format of the instruction is
  ```
  90        ABC123
  ```

- **91 Set up a numeric constant** - any number up to the maximum possible within 36 binary digits; negative numbers can be specified; format of the instruction is
  ```
  91        1234567
  ```

  Instructions of these two types are normally placed at the end of a stage.

- **92 Define a section letter for variables** - a bunch of variables can be referred to as, for example, A0-A19 - the dummy instruction 92 tells the program where A0 is to be stored in "core store"; subsequent variables in the A.. range follow on immediately after A0.
  `92   A   100` tells the program that A0 is located at register 100. The letters R, S, O and I cannot be used as sections. 92-instructions for all sections must be included at the start of the program.

- **93 Define stage storage** - specify the storage location on drum and in core store for each stage of the program.
  `93   320   1136   5   0` signifies that stage 5 is 16 blocks (64 instructions) long (0 = 16 blocks, 1-15 = 1-15 blocks), will be stored at address 1136 on drum, and wil be loaded into register 320 in core store. 93-instructions for all stages must be included at the start of the program.

- **94 Identify a new stage** - this instruction is the first of a new stage, and identifies the stage number in the c1-address
  ```
  94         3
  ```

- **96 End card** - indicates the end of a set of punched cards comprising one program. The c1-address indicates the first stage of the program to be executed (it will normally be, but need not be, stage 1).

Times for drum transfers, instructions 15 and 16:

Time = {head-switching + access + transfer}, where:

> **Head switching: for function 15 = 4,600 ± 900; for function 16 = 4250 ± 800 microseconds;**
> **Access time: average = 11.5 milliseconds (ie half a drum revolution);**
> **Transfer time: 23 millisec. per complete track, or 1.45 millisec. per 4-word block.**

Simon Lavington.