FIDO

What is FIDO?

FIDO simulates the operation of a processor.

What is the fetch execute cycle?

Program instructions are held in memory.

The processor fetches the next instruction, decodes it (works out which instruction it is), and executes it. The cycle repeats for the next instruction.

Which part does what?

Program instructions and data are stored in memory. Various registers are used to together so that instructions can be executed in the correct sequence.

What registers are there?

The instruction register (IR) holds *the current instruction*, after it has been fetched from memory and while it is decoded and executed.

The accumulator (ACC) holds a piece of data while arithmetic and logical operations are carried out.

The program counter (PC) is the *address* of the *next* instruction in memory. Usually after each instruction is executed, the PC is incremenetd so it points to the next instruction. For jump (JMP) instructions, the jump destination is just loaded into the PC.

The index register (IX) is used as a pointer into memory, for indexed addressing.

The memory address register (MAR) holds the address of a memory location, during a read or write operation.

The memory data register (MDR) holds data prior to it being written into memory, or it receives data from memory during a read.

The plus flag is set if the last compare instruction gives a positive result.

Th e zero flag is set if the last compare instruction gives a zero result.

The negative flag is set if the last compare instruction gives a negative result.

In real processors these flags (and others) are single bits in a 'program status word'. In Fido they are displayed separately.

Registers and memory locations are made of two-state electronic devices, so that the hold strings of 1s and 0s typically 32 or 64). When a value is written into a register or memory cell, the previous contents are over-written and lost.

What happens in a fetch?

The PC contains the address of the next instruction. This is copied into the MAR, through a bus internal to the processor - a data highway. A read operation is signalled to the memory, over the control bus which connects them. The MAR is placed on the address bus, which goes to memory.

The memory responds to the read signal (together with the clock signal also over the control bus) by retrieving the contents of the location given on the address bus. It places the result (which is the next instruction) on the data bus.

The processor copies the data bus into the MDR, and then through the internal bus to the IR.

At this point the next instruction has been fetched, and the fetch phase is complete. The processor will go on to decde the instruction, and execute it, which will often require further memory read or writes.

Instructions, opcodes, mnemonics and operands

An instruction is made up of an opcode and an operand. The opcode is the operation code, which instruction it is, and the operand is the data to use. The opcode is a binary string. Usually instead we use (in assembl language programming) a short word - a mnemonic. For example, ADI means add something to the accumulator. The operand is what number to add.

Both opcode and operand are actually binary strings. Instead we often use hex, or the mnemonic. For example

	opcode	operand
binary	1000	0011
hex	8	3
mnemonic	ADI	3

To keep things simple, FIDO uses base 10 instead of binary or hex. Real processors have instructions which are up to 32 or 64 bits long.

Addressing Modes

FIDO can use three addressing modes - immediate, direct and indexed.

In immediate addressing, the operand is the data. for example, ADI 3 means add 3 to the accumulator.

In direct addressing, the operand is the address of the data. For example, ADD 10 means add into the accumulator the number held at address 10. To carry this out, the processor has to read the memory again.

In indexed addressing, the address is the sum of the operand and the contents of the index register. So the instruction ADX 10 means, if IX contains 1, add to ACC the contents of address 11. If IX contains 2, it adds the contents of address 12.

T he addressing mode is shown by the letter I, D or X in the mnenomic. For examples, LDI means load immediate, LDD means load direct, and LDX is load indexed.

FIDO Instruction Set

LDI LDD LDX

Load the accumulator. Data is placed in the accumulator

SDD STX

Store accumulator. Contents of the accumulator are stored in memory

ADI ADD ADX

Add. Data is added to the accumulator

SBI SBD SBX

Subtract. Data is subtracted from the accumulator

CPI CPD CPX

Compare and set flags. Flags are set as if the data were subtracted from the accumulator. For example, if the accumulator contains 5, then CPI 4 would set the plus flag and clear the zero and neg flags.

JMP

Jump. Start executing from a new location.

JPG

Jump on Greater. Branch to a new location if the neg flag is set. For example, if the accumulator contains 5, then

CPI 6

JPG 10

will branch to address 10.

JPE

Jump on Equal. Branch to a new location if the zero flag is set. For example, if the accumulator contains 5, then

CPI 5

JPE 10

branches to address 10.

JPL

Jump on Less. Branch if the plus flag is set.

INP

Input a value from the keyboard into the accumulator. For a real processor, this might be a call to a low level sub-routine in ROM to read the keyboard, or a call to an operating system routine to do the same. It would not be a single instruction.

OUT

Output the contents of the accumulator. Like INP, this would be a ROM or OS routine call.

MVX

Move to IX register. Immediate addressing. MVX 10 puts 10 in the IX register

INX

Increment the IX register

DEX

Decrement the IX register

HALTING

If FIDO loads a blank instruction, it halts. Memory locations are blank by default.

This differs from real processors in two ways. Firstly real processors do not normally 'halt'. The fetch-execute cycle continues so long as power is supplied.

Secondly, memory locations canot be 'blank' or 'empty'. They are made of two-state devices each of which hold a 0 or 1. A location might contain a set of 0s, but this is not the same as blank.

Sample Programs

0: Add 3 and 4

```
0       LDI 3     // put 3 in accumulator - immediate addressing
1       ADI 4     // add 4 into accumulator - immediate addressing
2       OUT       // output accumulator = 7
```

1: Compare memory contents (10 and 11), output larger

```
0       LDD 10          // Get contents of address 10
1       CPD 11          // Compare with 11
2       JPG 5           // If 11 is greater, skip next part
3       OUT             // Output contents of 10
4                       // Halts
5       LDD 11          // Get contents of 11
6       OUT             // Output
10      1               // Data
11      2
```

2: Sum memory block

Use a loop and indexed addressing to add up the contents of a block of memory. The key instruction is ADX 10. This adds into the acc the contents of address 10+ix. The data block starts at 10, and we use ix as an offset beyond the block start.

```
0       LDI 0           // sum is in acc - initialise to 0
1       MVX 0           // initialise ix to 0
2       ADX 10          // add to acc contents of address 10+ix
3       INX             // increment ix
4       CXI 4           // compare ix with 4
5       JPG 2           // if 4 is geater, loop back to 2
6       OUT             // output sum
..
10      1               // data starts here
11      2
12      3
14      4
```

3: Input 5 numbers, output the largest

```
0       LDI 0           // initialise
1       STD 15          // count it stored at 15
2       STD 16          // maximum so far stored at 16
3       INP             // Input a number
4       CPD 16          // Compare with maximum so far
5       JPG 7           // If greater than this, skip next part
6       STD 16          // Store new maximum
7       LDD 15          // Get count
8       ADI 1           // Increment
9       STD 15          // Save count
10      CPI 5           // Compare
11      JPG 3           // If 5 > count, loop back
12      LDD 16          // Get maximum
13      OUT             // Output it
```

Comments

Send them to me, wm@waltermilner.com

Walter