DOG-1

Danny's Obtuse Gadget, version 1.0

(I'm writing this as if the thing was finished - far from it! - So this is all provisional design)

A very limited computer based on an Arduino Uno and a TM1638 I/O card. Runs as a virtual machine on top of the Arduino. Inspired by the single-board computers of the mid-1970's notably the [MK14](#) and [KIM-1](#). The architecture and instruction set is loosely derived from the processors used on these machines, the [SC/MP](#) and [6502](#) respectively.

Some novelty is offered by the hardware used is that it should be possible to interface with the built-in I/O of the Arduino. So eg. a digital thermometer could be made by attaching the appropriate sensor and programming the DOG-1 directly, without access to a PC.

Basic Architecture

- 16-bit addressing
- 8-bit instructions
- 8-bit data

Registers

Operations will be carried out using the following registers:

16-Bit

- Program Counter (PC) - steps through program
- X Register (XR) - an auxiliary register

8-Bit

- Accumulators A and B
- Status Register (SR) - system flags
- Stack Pointer (SP) - for remembering the origin of subroutine jumps

Note - I think I'll change this rather 6502-like setup after reading the 6800 datasheet. That has 2 accumulators and one X index register. Seems to make for a simpler but more versatile instruction set.

Also quite interested in trying stack-oriented programming, 'check Here is a list of several stack manipulation operators, including SWAP' https://www.forth.com/starting-forth/2-stack-manipulation-operators-arithmetic/ Also Stack Operations in 6800 doc.

Flags

Bit	Flag	Name	Description
0	N	Negative	Set if bit 7 of ACC is set
1	V	Overflow	-
2	Z	Zero	-
3	C	Carry	-

Addressing Modes

nn = 2 hex digits nnnn = 4 hex digits

Mode	Assembler Format	Description
Immediate	#nn	Value is given immediately after opcode
Absolute	nnnn	Value is contained in the given address
Indirect Absolute	(nnnn)	-

Absolute Indexed, X	nnnn, X	-
Absolute Indexed, Y	nnnn, Y	-
Relative	nnnn	-

I/O

TM1638 Card

	0	1	2	3	4	5	6	7
LEDs	*	*	*	*	*	*	*	*
7-Segs	8	8	8	8	8	8	8	8
Buttons	o	o	o	o	o	o	o	o

The LEDs will usually display the contents of the Status Register.

Generally, the 7-Segment displays 0-3 will display the current value (address) of the Program Counter. Displays 6 and 7 showing the contents at that address. All in hexadecimal.

The functionality of the I/O will depend on the system's mode : **Program** or **Run**. Display 4 shows the current mode, P or R.

Push-button 4 switches between these modes. At any time, pressing buttons 4 and 5 **together** will reset the PC to 0000.

Program Mode

Pressing the buttons 0-3, 6-7 will increment the value corresponding to that of the display above it. Programming is achieved by pressing button 3 to increment the PC (with overflow occurring, counting up on displays 0-2). Pressing button 7 will increment the value on display 7 (*without* overflowing to display 6), ditto for button 6/display 6, together providing the value at the given address.

Pressing button 5 will switch the response from increment to decrement. The PC buttons/display *does* carry values and wrap at max and min (0000). The code buttons/display act independently to each other and don't wrap in the <0 direction.

Pressing button 4 will switch to **Run** mode.

Run Mode

Initially the system will be halted at the current address. Pressing button 3 will single-step through the program (pressing buttons 0-3 will cause the PC to skip to the corresponding address [running or skipping code in between? TBD]).

Alternately the program may be run in real time by pressing button 5. Pressing this button again will halt the program.

The HALT opcode will terminate a program and wait for keyboard input before switching to Program mode and zeroing the program counter.

Clicking the first (0) and last (7) buttons simultaneously will **wipe all memory**.

Error Messages

xxxxnoPE - non-existent operation at xxxx

Instruction Set

note to self - things like LDA will have a version for each of the addressing modes, ~ 6, so it's probably an idea to hop 8 values between base versions...hmm, testing values for switch statements via masks?

Using index register - probably mainly for table lookup, maybe for subroutine-like things too should support ld, st, inc & dec, swap with PC, conditional swap

Ok, save long list until later, start with a subset...

Canonical version is in the code!!

Using the display etc. and Arduino I/O from code will need some specialised opcodes. Maybe : USE // to decouple device from system UNUSE PEEK & POKE - yes!!!! To set/get values on/from devices. Interrupt-driven bits?

Instruction	OpCode	Size	Flags	Description
NOP	00	1	-	No operation
CLS	01	1	all	clear flags
LDA #nn	10	2	-	Load acc A, immediate
LDA nnnn	11	3	-	Load acc A from value at address
STA nnnn	12	3	-	Store acc A at given address
LDB #nn	18	2	-	Load acc B, immediate
LDB nnnn	19	3	-	Load acc B from value at address
STB nnnn	20	3	-	Store acc B at given address
SPC nnnn	28	3	-	store program counter value at address
JMP nnnn	29	3	-	jump to given address
JNZ nnnn	30	3	-	jump to given address if zero flag set
AND	38	1	-	bitwise AND of acc A and acc B, result in A
OR	39	1	-	bitwise OR of acc A and acc B, result in A
XOR	40	1	-	bitwise XOR of acc A and acc , result in AB
NGA	41	1	-	invert bits in acc A
NGB	42	1	-	invert bits in acc B
RLA	43	1	C	rotate bits in acc A left + carry
RRA	44	1	C	rotate bits in acc A right + carry
RLB	45	1	C	rotate bits in acc B left + carry
RRB	46	1	C	rotate bits in acc B right + carry
ADD	50	1	C	ADD acc A and acc B, , result in A
SUB	51	1	NZ	subtract acc B from acc A, result in A
CMP	52	1	NZ	subtract acc B from acc A, discard result

Instruction	OpCode	Size	Operation	Description

NOP	00	1	PC++	No operation
CLF	00	1	PC++	No operation
JMP nnnn	20	3	PC <- nn	jump to given address
JNZ nnnn	20	3	PC <- nn	jump to given address
AND #nn	30	1	Halt	Stops program flow
AND nnnn	30	1	Halt	Stops program flow
OR #nn	30	1	Halt	Stops program flow
OR nnnn	30	1	Halt	Stops program flow
XOR #nn	30	1	Halt	Stops program flow
XOR nnnn	30	1	Halt	Stops program flow
ROL	30	1	Halt	Stops program flow
ROR	30	1	Halt	Stops program flow
ADD #nn	FF	1	Halt	Stops program flow
ADD nnnn	FF	1	Halt	Stops program flow
SUB #nn	FF	1	Halt	Stops program flow
SUB nn	FF	1	Halt	Stops program flow
CMP	FF	1	Halt	Stops program flow
HLT	FF	1	Halt	Stops program flow
HLT	FF	1	Halt	Stops program flow
HLT	FF	1	Halt	Stops program flow
HLT	FF	1	Halt	Stops program flow
HLT	FF	1	Halt	Stops program flow

Demo code

- 16x16bit multiply
- 16x16bit multiply
- Pseudorandom number generator

Games? Lunar Lander? noughts & crosses?

Using Arduino I/O - digital thermometer? beepy machine?

as in https://en.wikipedia.org/wiki/Linear-feedback_shift_register

See Also

This blog post gave me enough of how-to on interfacing with the TM1638 to get started:

https://blog.3d-logic.com/2015/01/10/using-a-tm1638-based-board-with-arduino/

See also :

http://obsolescence.wixsite.com/obsolescence/kim-uno-summary-c1uuh