despairVM architecture v2:

Memory Layout

There are 4 different types of memory spaces in despairVM and they are called Code Space, Data Space, Global Data Space, and Stack Space. Code Space is the only memory that contains executable data (opcodes) and others are not executable. Code Space cannot be changed at runtime, it is loaded when depairVM is booting. It is accessed only by the VM using Program Counter (Instruction Pointer) register. Data Space and Global Data Space is kind of same, they can be accessed and modified at runtime using cpu instructions. The only difference between the two is that there are cpu instructions that can access Global Data Space directly using the 32 bit address. But Data Space cannot be accessed directly. Lastly, Stack Space is a only accessed when you use push and pop instructions.

Registers

There are 256 64 bit integer registers named reg[0] … reg[255], and there are another 256 32 bit float registers named freg[0] … freg[255]

reg[254] and reg[255] have special purpose, reg[254] contains address to Global Data Space while reg[255] contains address to Data Space. This means replacing reg[255] with some other data will make the program running on VM never be able to access the Data Space.

Apart from those registers there are two special purpose registers: Program Counter and Stack Pointer.

GPU

GPU handles drawing to frame buffer. It can also perform things like image rotation, flipping, alpha blending etc.

Ports

The CPU communicates with different sub systems like Thread Manager, Storage Manager, GPU, DMA Manager etc via the ports. Here are the list of ports for different subsystems:

Port Number	**Description**
0	GPU Effects (1 byte) 7 - Reserved 6 - Reserved 5 - Reserved 4 - Only one color (no other effect are applied, useful for drawing fonts. If x and y isn't inside the frame buffer, it won't be drawn at all!) 3 - Flip Y 2 - Flip X 1 - Alpha Blending 0 – Rotation
1	Rotation in degree (used by GPU)
3	Frame Buffer 'In' DMA To initiate this DMA, write the pointer to

	memory that you want to be copied to frame buffer in this port.
4	Frame Buffer 'Out' DMA To initiate this DMA, write the pointer to memory that you want frame buffer to be copied to in this port.
5	Keyboard Port To find the status of a key, write the ASCII code of the key to this port and then read this port (1 = pressed and 0 = not pressed).
6	Make Heap To allocate heap memory, write the size (in bytes) of heap to this port and then read this port for pointer to newly created heap.
14	Destroy Heap To destroy heap memory, write the pointer to the heap memory here.
15	File Stream Object Read/Write file stream object here.
23	File Input/Output 1 Any input to/output from storage manager will be here.
31	File Input/Output 2 Any input to/output from storage manager will be here.
39	File Command Write commands to storage manager here.
40	String Object Read/Write string object here.
48	String Input/Output 1 Any input to/output from string manager will be here.
56	String Input/Output 2 Any input to/output from string manager will be here.
64	String Command Write commands to string manager here.
65	Create Thread Write code space address (entry point of thread) to this port to create a new thread.
66	DMA Address 1 Source DMA address
74	DMA Address 2 Destination DMA address
82	DMA Size

	DMA transfer size. Writing to this port will initiate DMA transfer.
83	Thread Parameter Thread parameter pointer here. If pointer is 0, no parameter is passed.
91	GPU Command Writing 0 in most significant byte (32 bits value) will clear the frame buffer with background color (RGB) in rest of the bits (24 bits) . Writing anything but 0 in most significant byte and writing 0 in rest of the 24 bits will present the buffer to the screen. The monochrome drawing reads color from this port. So, Write 1 in most significant byte and write RGB value in rest of the 24 bits for that purpose. (port 0, bit 4)

File Manager (Storage Manager) commands

Command value	**Description**
0	Creates new file stream object and puts its address in 'File Stream Object' port.
1	Destroys file stream object in 'File Stream Object' port.
2	Gets the size of the file stream object in 'File Stream Object' port and puts the size in 'File Input/Output 1' port.
3	Sets the file pointer position of the file stream object in 'File Stream Object' port to the offset in 'File Input/Output 1' port.
4	Opens a file as text file. Path should be written in 'File Input/Output 1 and the file stream object should be kept in 'File Stream Object' port. Boolean return value is kept in 'File Input/Output 2', which tells if the file was successfully opened or not (1 = success, and 0 = fail)
5	Opens a file as binary file. Path should be written in 'File Input/Output 1' port and the file stream object should be kept in 'File Stream Object' port. Boolean return value is kept in 'File Input/Output 2', which tells if the file was successfully opened or not (1 = success, and 0 = fail)
6	Closes the file. File stream object should be in

	'File Stream Object'.
7	Reads the file as binary file. File stream object should be in 'File Stream Object', address to buffer should be in 'File Input/Output 1' port, and size in bytes in 'File Input/Output 2'.
8	Reads the file as text file. File stream object should be in 'File Stream Object', address to string object should be in 'File Input/Output 1' port, and size in lines in 'File Input/Output 2' (if 0 reads till the end of file).
9	Writes the file as binary file. File stream object should be in 'File Stream Object', address to buffer should be in 'File Input/Output 1' port, and size in bytes in 'File Input/Output 2'.
10	Writes the file as text file. File stream object should be in 'File Stream Object' and address to string should be in 'File Input/Output 1' port.

String Manager commands

Command value	**Description**
0	Create String object. Newly created string object can be read from 'String Object' port.
1	Destroy String object. String object to be deleted should be kept on 'String Object' port.
2	Get String size. String Object should be kept on 'String Object' port and the size can be read from 'String Input/Output 1' port.
3	Get char at index. String Object should be kept on 'String Object' port, char can be read from 'String Input/Output 1' port, and the index should be written to 'String Input/Output 2' port.
4	Set char at index. String Object should be kept on 'String Object' port, char should be kept on 'String Input/Output 1' port, and the index should be written to 'String Input/Output 2' port.
5	Append char array. String Object should be kept on 'String Object' port and the char array should be kept on 'String Input/Output 1' port.
6	Append String. String Object should be kept on 'String Object' port and the string source should be kept on 'String Input/Output 1' port.
7	Append Integer. String Object should be kept on 'String Object' port and the integer should be kept on 'String Input/Output 1' port.

8	Append Float. String Object should be kept on 'String Object' port, float should be kept on 'String Input/Output 1' port (Low DWORD), and precision should be kept on 'String Input/Output 2' port (Low DWORD).
9	Clear String. String Object should be kept on 'String Object' port.
10	Get Char Array. String Object should be kept on 'String Object' port, and char array can be read from 'String Input/Output 1' port.
11	Compare string objects in 'String Object' port and 'String Input/Output 1' port. Result can be read from 'String Input/Output 2' port (1 = equal and 0 = not equal).
12	Copy string. String Object should be kept on 'String Object' port and the string source should be kept on 'String Input/Output 1' port.

Executable File

The file that contains the binary data which can be executed by the VM is the executable file. Executable files should have the extension '.dbin'. Here is the contents of the executable file header:

Offset	**Description**	**Size**
0	Magic Number ('DPVM' in big endian)	4 bytes
4	Version of this executable file	4 bytes
8	Size of this header	4 bytes
12	Size of the code	4 bytes
16	Execution entry of the code	4 bytes
20	Data Space size	4 bytes
24	Stack Space size	4 bytes
28	Global Data Space size	4 bytes
32	Frame Buffer width	4 bytes
36	Frame Buffer height	4 bytes
40	Code Signature (SHA-256)	4 bytes
72	Global Pre Data	variable

Right after 'Global Pre Data', the binary code is stored.

CPU Opcode List

Glossary:

R	Integer Register
M	Integer Memory. Address in Global Data Space
[R]	Integer memory address in register
IMMI	Immediate integer value (32 bit unless the size is given, eg IMMI64 is 64 bit)
FR	Float Register
F-M	Float Memory. Address in Global Data Space
[F-R]	Float memory address in register
FIMMI	Immediate 32 bit float value
BM	8 bit value in memory. Address in Global Data Space
[BR]	8 bit value memory address in register

Here's the list of all the CPU instuction along with their hexadecimal values:

MOV:

0000 R R: MOV R, R

Moves from Register to Register.

0001 R R IMMI: MOV [R + IMMI], R

Moves from register 2 to address pointed by the Register 1 + IMMI.

0002 R R IMMI: MOV R, [R + IMMI]

Moves from address pointed by the Register 2 + IMMI to register 1.

0003 R R IMMI IMMI: MOV [R + IMMI], [R + IMMI]

Moves from address pointed by the Register 2 + IMMI to address pointed by the Register 1 + IMMI.

0004 R IMMI: MOV R, IMMI

Moves immediate to address pointed by the Register.

0005 R IMMI IMMI: MOV [R + IMMI], IMMI

Moves immidiate to address pointed by Register + IMMI.

0006 R M: MOV R, M

Moves from memory to Register.

0007 R M: MOV M, R

Moves from Register to memory.

0008 M M: MOV M, M

Moves from Memory to Memory.

0009 R R: MOV [R], R

Moves from register 2 to address pointed by the Register 1.

000a R R: MOV R, [R]

Moves from address pointed by the Register 2 to register 1.

000b R M: MOV [R], M

Moves from memory to address pointed by the Register.

000c R M: MOV M, [R]

Moves from address pointed by the Register to memory.

000d R R: MOV [R], [R]

Moves from address pointed by the Register 2 to address pointed by the Register 1.

000e M IMMI: MOV M, IMMI

Moves immidiate to memory.

000f R IMMI: MOV [R], IMMI

Moves immidiate to address pointed by Register.

ADD:

0010 R R: ADD R, R

Adds from Register to Register.

0011 R IMMI: ADD R, IMMI

ADDs immediate to address pointed by the Register.

SUB:

0012 R R: SUB R, R

Subs from Register to Register.

0013 R IMMI: SUB R, IMMI

SUBs immediate to address pointed by the Register.

MUL:

0014 R R: MUL R, R

Muls from Register to Register.

0015 R IMMI: MUL R, IMMI

MULs immediate to address pointed by the Register.

DIV:

0016 R R: DIV R, R

Divs from Register to Register.

0017 R IMMI: DIV R, IMMI

DIVs immediate to address pointed by the Register.

MOD:

0018 R R: MOD R, R

Mods from Register to Register.

0019 R IMMI: MOD R, IMMI

MODs immediate to address pointed by the Register.

AND:

001a R R: AND R, R

ANDs from Register to Register.

001b R IMMI: AND R, IMMI

ANDs immediate to address pointed by the Register.

OR:

001c R R: OR R, R

ORs from Register to Register.

001d R IMMI: OR R, IMMI

ORs immediate to address pointed by the Register.

XOR:

001e R R: XOR R, R

XORs from Register to Register.

001f R IMMI: XOR R, IMMI

XORs immediate to address pointed by the Register.

SHL:

0020 R IMMI8: SHL R, IMMI8

Shifts Register left by IMMI8 bits.

0021 R R: SHL R, R

Shifts Register left by Register bits.

SHR:

0022 R IMMI: SHR R, IMMI8

Shifts Register right by IMMI8 bits.

0023 R R: SHR R, R

Shifts Register right by Register bits.

NOP:

0024 NOP

No operation.

JMP:

0025 IMMI JMP IMMI

Jumps to the immidiate address.

0026 IMMI JMPR IMMI

Jumps to the immidiate address (relative).

0027 R IMMI JC R, IMMI

Condional jump. Jump if value in register is 0.

0028 R IMMI JCR R, IMMI

Condional jump. Jump if value in register is 0. (relative).

MOVP Pointer:

0029 R R IMMI MOVP R, [R + IMMI]

Moves 64 bits pointer from address pointed by register + IMMI to register.

002a R R IMMI MOVP [R + IMMI], R

Moves 64 bits pointer from register to address pointed by register + IMMI.

002b R R IMMI IMMI MOVP [R + IMMI], [R + IMMI]

Moves 64 bits pointer from address pointed by the Register 2 + IMMI to address pointed by the Register 1 + IMMI.

002c R M MOVP R, M

Moves 64 bits pointer from memory to Register.

002d R M MOVP M, R

Moves 64 bits pointer from register to memory.

002e R R MOVP R, [R]

Moves 64 bits pointer from address pointed by register to register.

002f R R MOVP [R], R

Moves 64 bits pointer from register to address pointed by register.

CALL:

0030 IMMI CALL IMMI

Push program counter to stack and jump to memory in codespace.

RET:

0031 RET

Pop program counter from stack and jump to that position.

PUSH:

0032 R PUSH R

Push register (64 bits) to stack.

POP:

0033 R POP R

Pop register (64 bits) from stack.

DRW:

0034 R R MR DRW R, R, [MR]

Draw on screen. Coordinates (R1, R2) and image pointed by R3.

OUT:

0035 R IMMI OUT R, IMMI8

Output IMMI to port address in register.

0036 R IMMI OUT R, IMMI16

Output IMMI to port address in register.

0037 R IMMI OUT R, IMMI32

Output IMMI to port address in register.

0038 R IMMI OUT R, IMMI64

Output IMMI to port address in register.

0039 R IMMI OUT IMMI, R8

Output register to port address in IMMI.

003a R IMMI OUT IMMI, R16

Output register to port address in IMMI.

003b R IMMI OUT IMMI, R32

Output register to port address in IMMI.

003c R IMMI OUT IMMI, R64

Output register to port address in IMMI.

003d R R OUT R, R8

Output register 2 to port address in register 1.

003e R R OUT R, R16

Output register 2 to port address in register 1.

003f R R OUT R, R32

Output register 2 to port address in register 1.

0040 R R OUT R, R64

Output register 2 to port address in register 1.

0041 IMMI IMMI OUT IMMI, IMMI8

Output IMMI 2 to port address in IMMI 1.

0042 IMMI IMMI OUT IMMI, IMMI16

Output IMMI 2 to port address in IMMI 1.

0043 IMMI IMMI OUT IMMI, IMMI32

Output IMMI 2 to port address in IMMI 1.

0044 IMMI IMMI OUT IMMI, IMMI64

Output IMMI 2 to port address in IMMI 1.

IN:

0045 R IMMI IN R8, IMMI
Input from port IMMI to register.

0046 R IMMI IN R16, IMMI
Input from port IMMI to register.

0047 R IMMI IN R32, IMMI
Input from port IMMI to register.

0048 R IMMI IN R64, IMMI
Input from port IMMI to register.

0049 R R IN R8, R
Input from port register 2 to register 1.

004a R R IN R16, R
Input from port register 2 to register 1.

004b R R IN R32, R
Input from port register 2 to register 1.

004c R R IN R64, R
Input from port register 2 to register 1.

FCON:

004d R FR FCON R, FR
Converts floating point in float register to integer and moves it to register.

004e R FR FCON FR, R

Converts integer in register to floating point number and moves it to float register.

004f FR M FCON FR, IMMI

Converts immidiate integer to floating point number and moves it to float register.

0050 R FIMMI FCON R, FIMMI

Converts immidiate float to integer number and moves it to register.

FMOV:

0051 FR FR FMOV FR, FR

Moves float register 2 to float register 1.

0052 R FR IMMI FMOV FR, [F-R + IMMI]

Moves float memory pointed by the register + IMMI to float register.

0053 R FR IMMI FMOV [F-R + IMMI], FR

Moves float register to float memory pointed by the register + IMMI.

0054 R FR IMMI IMMI FMOV [F-R + IMMI], [F-R + IMMI]

Moves float memory pointed by the register 2 + IMMI to float memory pointed by the register 1 + IMMI.

0055 R FR FMOV FR, FIMMI

Moves immidiate floating point number to float register.

0056 R FR IMMI FMOV [F-R + IMMI], FIMMI

Moves immidiate floating point number to float memory pointed by register + IMMI.

0057 FR M FMOV FR, F-M

Moves float memory to float register.

0058 FR M FMOV F-M, FR

Moves float register to float memory.

0059 R FR FMOV FR, [F-R]

Moves float memory pointed by the register to float register.

005a R FR FMOV [F-R], FR

Moves float register to float memory pointed by the register.

005b R FR FMOV F-M, [F-R]

Moves float memory pointed by the register to float memory.

005c R FR FMOV [F-R], F-M

Moves float memory to float memory pointed by the register.

005d R FR FMOV [F-R], [F-R]

Moves float memory pointed by the register 2 to float memory pointed by the register 1.

005e R FR FMOV F-M, F-M

Moves float memory 2 to float memory 1.

005f R FR FMOV F-M, FIMMI

Moves immidiate floating point number to float memory.

0060 R FR FMOV [F-R], FIMMI

Moves immidiate floating point number to float memory pointed by register.

FADD:

0061 FR FR FADD FR, FR

Adds float register 2 to float register 1 and puts result in float register 1.

0062 R FR FADD R, FR

Adds float register to register and moves result to register.

0063 R FR FADD FR, R

Adds register to float register and moves result to float register.

0064 R FR FADD FR, FIMMI

Adds immidiate floating point number to float register and moves result to float register.

0065 R FR FADD R, FIMMI

Adds immidiate floating point number to register and moves result to register.

FSUB:

0066 FR FR FSUB FR, FR

Subs float register 2 from float register 1 and puts result in float register 1.

0067 R FR FSUB R, FR

Subs float register from register and moves result to register.

0068 R FR FSUB FR, R

Subs register from float register and moves result to float register.

0069 R FR FSUB FR, FIMMI

Subs immidiate floating point number from float register and moves result to float register.

006a R FR FSUB R, FIMMI

Subs immidiate floating point number from register and moves result to register.

FMUL:

006b FR FR FMUL FR, FR

Muls float register 2 with float register 1 and puts result in float register 1.

006c R FR FMUL R, FR

Muls float register with register and moves result to register.

006d R FR FMUL FR, R

Muls register with float register and moves result to float register.

006e R FR FMUL FR, FIMMI

Muls immidiate floating point number with float register and moves result to float register.

006f R FR FMUL R, FIMMI

Muls immidiate floating point number with register and moves result to register.

FDIV:

0070 FR FR FDIV FR, FR

Divs float register 1 by float register 2 and puts result in float register 1.

0071 R FR FDIV R, FR

Divs register by float register and moves result to register.

0072 R FR FDIV FR, R

Divs float register by register and moves result to float register.

0073 R FR FDIV FR, FIMMI

Divs float register by immidiate floating point number and moves result to float register.

0074 R FR FDIV R, FIMMI

Divs register by immidiate floating point number and moves result to register.

FMOD:

0075 FR FR FMOD FR, FR

Mods float register 1 by float register 2 and puts result in float register 1.

0076 R FR FMOD R, FR

Mods register by float register and moves result to register.

0077 R FR FMOD FR, R

Mods float register by register and moves result to float register.

0078 R FR FMOD FR, FIMMI

Mods float register by immidiate floating point number and moves result to float register.

0079 R FR FMOD R, FIMMI

Mods register by immidiate floating point number and moves result to register.

8 Bit Movs:

007a R R IMMI BMOV R, [BR + IMMI]

Moves 8 bit data from byte memory pointed by register 2 + IMMI to register 1.

007b R R IMMI BMOV [BR + IMMI], R

Moves 8 bit data from register 2 to memory pointed by register 1 + IMMI.

007c R R IMMI IMMI BMOV [BR + IMMI], [BR + IMMI]

Moves 8 bit data from memory pointed by register 2 + IMMI to memory pointed by register 1 + IMMI.

007d R IMMI IMMI8 BMOV [BR + IMMI], IMMI8

Moves immidiate 8 bit data to byte memory pointed by register + IMMI.

007e R M BMOV R, BM

Moves 8 bit data from byte memory to register.

007f R M BMOV BM, R

Moves 8 bit data from register to byte memory.

0080 R R BMOV R, [BR]

Moves 8 bit data from byte memory pointed by register 2 to register 1.

0081 R R BMOV [BR], R

Moves 8 bit data from register 2 to memory pointed by register 1.

0082 R R BMOV [BR], [BR]

Moves 8 bit data from memory pointed by register 2 to memory pointed by register 1.

0083 R R BMOV BM, BM

Moves 8 bit data from byte memory 2 to byte memory 1.

0084 R IMMI8 BMOV [BR], IMMI8

Moves immediate 8 bit data to byte memory pointed by register.

0085 M IMMI8 BMOV BM, IMMI8

Moves immediate 8 bit data to byte memory.

Compare:

0086 R R CMPE R, R

Compares register 1 with register 2. If they are equal, register 1 equals 1 else 0.

0087 R R CMPNE R, R

Compares register 1 with register 2. If they are not equal, register 1 equals 1 else 0.

0088 R R CMPG R, R

Compares register 1 with register 2. If register 1 is greater, register 1 equals 1 else 0.

0089 R R CMPL R, R

Compares register 1 with register 2. If register 1 is less, register 1 equals 1 else 0.

008a R R CMPGE R, R

Compares register 1 with register 2. If register 1 is greater or equal, register 1 equals 1 else 0.

008b R R CMPLE R, R

Compares register 1 with register 2. If register 1 is less or equal, register 1 equals 1 else 0.

008c FR FR FCMPE R, FR, FR

Compares float register 1 with float register 2. If they are equal, register 1 equals 1 else 0.

008d FR FR FCMPNE R, FR, FR

Compares float register 1 with float register 2. If they are not equal, register 1 equals 1 else 0.

008e FR FR FCMPG R, FR, FR

Compares float register 1 with float register 2. If register 1 is greater, register 1 equals 1 else 0.

008f FR FR FCMPL R, FR, FR

Compares float register 1 with float register 2. If register 1 is less, register 1 equals 1 else 0.

0090 FR FR FCMPGE R, FR, FR

Compares float register 1 with float register 2. If register 1 is greater or equal, register 1 equals 1 else 0.

0091 FR FR FCMPLE R, FR, FR

Compares float register 1 with float register 2. If register 1 is less or equal, register 1 equals 1 else 0.

Float Out/In:

0092 FR IMMI FOUT IMMI, FR

Put floating point number in port number in IMMI.

0093 FR IMMI FIN FR, IMMI

Gets floating point number from port number in IMMI.

0094 IMMI FIMMI FOUT IMMI, FIMMI

Puts floating point immediate value in port number in IMMI

PUSHES and POPS

0095 R R PUSHES R, R

Pushes register (64 bit) R1 to R2 into stack.

0096 R R POPS R, R

Pops register (64 bit) R1 to R2 from stack.

Float Stack push and pop:

0097 FR FR FPUSHES FR, FR

Pushes float register FR1 to FR2 into stack.

0098 FR FR FPOPS FR, FR

Pops float register FR1 to FR2 from stack.

0099 FR FPUSH FR

Pushes float register into stack.

009a FR FPOP FR

Pops float register from stack.

Immediate Compares:

009b R IMMI CMPE R, IMMI

Compares register with register immi. If they are equal, register equals 1 else 0.

009c R IMMI CMPNE R, IMMI

Compares register with register immi. If they are not equal, register equals 1 else 0.

009d R IMMI CMPG R, IMMI

Compares register with register immi. If they are greater, register equals 1 else 0.

009e R IMMI CMPGE R, IMMI

Compares register with register immi. If they are greater or equal, register equals 1 else 0.

009f R IMMI CMPL R, IMMI

Compares register with register immi. If they are less, register equals 1 else 0.

00a0 R IMMI CMPLE R, IMMI

Compares register with register immi. If they are less or equal, register equals 1 else 0.

Timer opcode:

00a1 TIME

Puts milliseconds time in register 0.

00a2 SLEEP

Thread sleeps for a millisecond.

Random Number:

00a3 RAND

Generates a random number and puts it in register 0.