| | | | | |
|---|---|---|---|---|
| 00 | -ld r1,r2 | -ld r1,[r2] | -ld r1,[r2+int(nn)] | -ld r1,[r2+nnnn] |
| 01 | -xchg r1,r2 | -ld r1,byte[r2] | -ld r1,byte[r2+nn] | -ld r1,byte[r2+nnnn] |
| 02 | -xchg r1,[r2] | -ld r1,word[r2] | -ld r1,word[r2+nn] | -ld r1,word[r2+nnnn] |
| 03 | -ld DSB/ESB, PDB[r1] | -ld r1,short[r2] | -ld r1,short[r2+nn] | -ld r1,short[r2+nnnn] |
| 04 | -ld r1,sr0x | -ld [r2],r1 | -ld [r2+nn],r1 | -ld [r2+nnnn],r1 |
| 05 | -ld sr0x,r1 | -ld byte[r2],r1 | -ld byte[r2+nn],r1 | -ld byte[r2+nnnn],r1 |
| 06 | -ld r1,sr1x | -ld word[r2],r1 | -ld word[r2+nn],r1 | -ld word[r2+nnnn],r1 |
| 07 | -ld sr1x,r1 | -ld r1,int[r2] | -ld r1,int[r2+nn] | -ld r1,int[r2+nnnn] |
| 08 | -neg r1,r2 | -ld r1,[es:r2] | -ld r1,[es:r2+nn] | -ld r1,[es:r2+nnnn] |
| 09 | -sar r1,r2 | -ld r1,byte[es:r2] | -ld r1,byte[es:r2+nn] | -ld r1,byte[es:r2+nnnn] |
| 0a | -shl r1,r2 | -ld r1,word[es:r2] | -ld r1,word[es:r2+nn] | -ld r1,word[es:r2+nnnn] |
| 0b | -shr r1,r2 | -ld r1,short[es:r2] | -ld r1,short[es:r2+nn] | -ld r1,short[es:r2+nnnn] |
| 0c | -rol r1,r2 | -ld [es:r2],r1 | -ld [es:r2+nn],r1 | -ld [es:r2+nnnn],r1 |
| 0d | -ror r1,r2 | -ld byte[es:r2],r1 | -ld byte[es:r2+nn],r1 | -ld byte[es:r2+nnnn],r1 |
| 0e | -rcl r1,r2 | -ld word[es:r2],r1 | -ld word[es:r2+nn],r1 | -ld word[es:r2+nnnn],r1 |
| 0f | -rcr r1,r2 | -ld r1,int[es:r2] | -ld r1,int[es:r2+nn] | -ld r1,int[es:r2+nnnn] |
| 10 | -add r1,r2 | -add [r2],r1 | -add [r2+nn],r1 | -add [r2+nnnn],r1 |
| 11 | -padd r1,r2 | -add r1,[r2] | -add r1,[r2+nn] | -add r1,[r2+nnnn] |
| 12 | -adc r1,r2 | -adc [r2],r1 | -adc [r2+nn],r1 | -adc [r2+nnnn],r1 |
| 13 | -padds r1,r2 | -adc r1,[r2] | -adc r1,[r2+nn] | -adc r1,[r2+nnnn] |
| 14 | -sub r1,r2 | -sub [r2],r1 | -sub [r2+nn],r1 | -sub [r2+nnnn],r1 |
| 15 | -psub r1,r2 | -sub r1,[r2] | -sub r1,[r2+nn] | -sub r1,[r2+nnnn] |
| 16 | -sbc r1,r2 | -sbc [r2],r1 | -sbc [r2+nn],r1 | -sbc [r2+nnnn],r1 |
| 17 | -psubs r1,r2 | -sbc r1,[r2] | -sbc r1,[r2+nn] | -sbc r1,[r2+nnnn] |
| 18 | -and r1,r2 | -and [r2],r1 | -and [r2+nn],r1 | -and [r2+nnnn],r1 |
| 19 | -piadds r1,r2 | -and r1,[r2] | -and r1,[r2+nn] | -and r1,[r2+nnnn] |
| 1a | -or r1,r2 | -or [r2],r1 | -or [r2+nn],r1 | -or [r2+nnnn],r1 |
| 1b | -ppaadd r1,r2 | -or r1,[r2] | -or r1,[r2+nn] | -or r1,[r2+nnnn] |
| 1c | -xor r1,r2 | -xor [r2],r1 | -xor [r2+nn],r1 | -xor [r2+nnnn],r1 |
| 1d | -pisubbs r1,r2 | -xor r1,[r2] | -xor r1,[r2+nn] | -xor r1,[r2+nnnn] |
| 1e | -test r1,r2 | -test [r2],r1 | -test [r2+nn],r1 | -test [r2+nnnn],r1 |
| 1f | -ppradd r1,r2 | -test r1,[r2] | -test r1,[r2+nn] | -test r1,[r2+nnnn] |
| 20 | -cmp r1,r2 | -cmp [r2],r1 | -cmp [r2+nn],r1 | -cmp [r2+nnnn],r1 |
| 21 | -imul64 r1,r2 | -cmp r1,[r2] | -cmp r1,[r2+nn] | -cmp r1,[r2+nnnn] |
| 22 | -mul r1,r2 | -mul [r2],r1 | -mul [r2+nn],r1 | -mul [r2+nnnn],r1 |
| 23 | -mul64 r1,r2 | -mul r1,[r2] | -mul r1,[r2+nn] | -mul r1,[r2+nnnn] |
| 24 | -div r1,r2 | -div [r2],r1 | -div [r2+nn],r1 | -div [r2+nnnn],r1 |
| 25 | -sort r1,r2 | -div r1,[r2] | -div r1,[r2+nn] | -div r1,[r2+nnnn] |
| 26 | | | | |
| 27 | | | | |
| 28 | | -ld cond r1,0 | -ld cond r1,int(nn) | -ld cond r1,nnnn |
| 29 | | -ppop r1; rotate r1,1 | ppop nn; rotate r1,nn | ppop nnnn; rotate r1,nnnn |
| 2a | | | -add,sub,… r1,nn | -add,sub,… r1,nnnn |
| 2b | | | -padd,psub,… r1,nn | -padd,psub,… r1,nnnn |
| 2c | | | | |
| 2d | -inc,inc2,inc4,…dec r1 | -inc,inc2,inc4,…dec [r1] | -inc,inc2,…dec [r1+nn] | -inc,…dec [r1+nnnn] |
| 2e | -swp4321,0001… r1 | -swp4321,0001… [r1] | -swp4321,0001… [r1+nn] | -swp4321,00… [r1+nnnn] |
| 2f | -swp1243,1111… r1 | -swp1243,1111… [r1] | -swp1243,1111… [r1+nn] | -swp1243,11… [r1+nnnn] |
| 30 | -fld q1,q2 | -fld q1,single[r1] | -fld q1,single[r1+nn] | -fld q1,single[r1+nnnn] |
| 31 | | -fld q1,double[r1] | -fld q1,double[r1+nn] | -fld q1,double[r1+nnnn] |
| 32 | -fld q1,PI,e,lg2… | -fld q1,extended[r1] | -fld q1,extended[r1+nn] | -fld q1,extend[r1+nnnn] |
| 33 | -fsin,fcos… | -fld q1,r1 | -fld q1,int(nn) | -flq q1,long(nnnn) |
| 34 | -fadd q1,q2 | -fld single[r1],q1 | -fld single[r1+nn],q1 | -fld single[r1+nnnn],q1 |
| 35 | -fsub q1,q2 | -fld double[r1],q1 | -fld double[r1+nn],q1 | -fld double[r1+nnnn],q1 |
| 36 | -fmul q1,q2 | -fld extended[r1],q1 | -fld extended[r1+nn],q1 | -fld extend[r1+nnnn],q1 |
| 37 | -fdiv q1,q2 | -fld r1,round(q1) | | -flq q1,single(nnnn) |
| 38 | -jbk cond,2n | -jmp cond,r1 | -jmp cond1, cond2, nn | -jmp cond1, cond2,nnnn |
| 39 | -skip cond,2n | -jr cond,r1 | -jr cond1, cond2, int(nn) | -jr cond1, cond2,nnnn |
| 3a | -ret cond1,cond2 | -call cond,r1 | -call cond1, cond2, nn | -call cond1, cond2,nnnn |
| 3b | | -calr cond,r1 | -calr cond1, cd2, int(nn) | -calr cond1, cond2,nnnn |
| 3c | | -jmp cond,[r1] | -jmp cond,[r1+int(nn)] | -jmp cond,[r1+nnnn] |
| 3d | | -call cond,[r1] | -call cond,[r1+int(nn)] | -call cond,[r1+nnnn] |
| 3e | INT n | -int n,ax=0 | -int n,ax=int(nn) | -int n,ax=nnnn |
| 3f | SIMD | SIMD | SIMD | SIMD |

| Den1 | Den2 | Description |
|---|---|---|
| SR0 | SP | Application Stack Pointer |
| SR1 | Flags | Flags |
| SR2 | PC or IP | Application Program Counter |
| SR3 | ITB | Interrupt Table Base |
| SR4 | PBT | Page Base Table |
| SR5 | PBD | Page Base Table Dimension |
| SR6 | DSB | Default Segment Base |
| SR7 | ESB | Extended Segment Base |
| | | |
| SRe | loopra | |
| SRf | looprb | |
| SR10 | S_SP | System Stack Pointer |
| SR11 | S_Flags | Flags |
| SR12 | S_PC | System Program Counter |
| SR13 | S_ITB | System Interrupt Table Base |
| SR14 | S_PBT | System Page Base Table |
| SR15 | S_PBD | System Page Base Table Dimension |
| SR16 | S_DSB | Default Segment Base |
| SR17 | S_ESB | Extended Segment Base |
| | | |
| SR1e | s_loopra | |
| SR1f | s_looprb | |

Implemented but not tested
Implemented but I don't expect it to work
Implemented and testen on at least one test case

**NOU:**
jump-uri cu 2 conditii
stiva creste in sus
[SP] contine ultima valoare salvata pe stiva
DSB,ESB,S_DSB,S_ESB contin echivalentul Page Directory Base de la Intel : adrese in memoria fizica
O valoare 0 inseamna acces direct la memoria fizica
PBT,S_PBT : adrese in memoria fizica unde se afla un tabel de dimensiune PBD respectiv S_PBD,
care contin date ce se pot incarca in DSB, ESB
(pentru o eventuala implementare hardware)
O aplicatie poate scrie registrii: SP, SPL, PC, ITB
DSB, ESB pot fi incarcati cu valori din tabela PBT => o aplicatie poate face switch intre mai multe paginari fara sa cheme OSul

Daca-mi poti sugera alte denumiri pentru registrii ...

Procesorul va avea 16 registrii de 32 bitzi r0,r2,r3,… sau ax,bx,cx,dx,ex… (ma gandesc sa fie permise ambele denumiri) Si 16 registrii floating point: q0,q2,…q15 Instructiunile vor fi:
2 bytes codul instructiunii
ex.
58 9a 11 12 = mov r5,short[r8+1211h]

vor exista doi registri de flag-uri a cate 8 bitzi
{initial ma gandeam la 8 registri , merge in hard dar la emulare sux} sunt definite 3 flag-uri : Carry,Zero,Overflow)
o operatiie care afecteaza flag-urile modifica tot registrul de 8 biti. Un registru tine ultimele flag-uri, celalalt pe cele de la penultima operatie.
Conditiile de la ld,jmp, etc pot fi : **-,c, nc, o, no, z, nz, loopcx,-, c2, nc2 ,o2 ,no2 ,z2 ,nz2,loopdx**
**loopc** – decrementeaza CX sau DX , nu afecteaza flag-urile

Daca primul byte este
**0..127** : instructiunea are 2 octeti
**128..191** :2 octeti codul instructiunii + 2 octeti de date
**192..255** :2 octeti codul instructiunii + 4 octeti de date
in plus **128..191** si **192..255** sunt la fel cu **64..127** cu exceptia operandului
**56..63** clasa speciala de instructiuni, adresa instructiunii urmatoare trebuie recalculata
FAULT means immediate termination of application

**int n,ax=nn**
set ax to nn then int n

**int n** – generate interrupt
if IBT=0 then call software_interrupt_from_application[n]
if (IBT<>0) and (dword ptr[IBT+n*4]<>0) then call dword ptr[IBT+n*4]

system interrupt:
system interrupt table:
  dword software_interrupts_from_system[256]
  dword software_interrupts_from_application[256]
  dword hardware_interrupts_from_external_devices[256]
  dword reserved[256] (exceptions?)

**conditional jumps and calls**
invalid addresses, stack overflow can cause a FAULT no matter whether the condition is true or false
for instructions with two conditions both must be true to execute
loop conditions in double condition instructions always decrement the loop register
the same loop condition repeated decrement the loop register only once
relative jumps are computed using the value of PC before the execution of the instruction
the assembler should accept a generic JMP and produce the shortest form

**ld r1,r2; xchg r1,[r2];**
r1=high nibble; r2=low nibble

**ld r1,srxx; ld srxx,r1**
special register=high nibble
**ld r1,sr0x / ld sr0x,r1**
load general use register with register x from current special register set (i.e. application or system)
**ld r1,sr1x / ld sr1x,r1** (privileged)
load general use register with register x from application special register set in system mode
undefined result in application mode – doesn't generate a FAULT

**ld r1,xxx[r2]; ld xxx[r2],r1; add,sub …**
r1=high nibble; r2=low nibble

**incn r1; decn r1; incn [r1]; decn [r1]**
adds or substracts 1,2,4,8,10,12,16,24 to r1 or [r1] and sets the **zero** and **carry** flags accordingly

**jmp cond,r1**
cond=high nibble; r1=low nibble

**jbk; skip**
high nibble=condition; low nibble=$-2,$-4,…; $+4;$+8;…

**swap**
0000,0004,0034,0234, 4321,0001,0012,0123, 0204,0103,0102,0304, 2040,1030,1020,3040
1243,2134,2143,4231, 4123,3412,2341,1324, 1111,2222,3333,4444, 3214,0404,ss34,sss4

**float**
fsin fcos ftan fsincos
fexp fln fround ftrunc
fsqrt finv fabs fminus
fpatan fprem

fld q1,fc_zero/fc_one/fc_two/fc_pi/fc_e/fc_ln2/fc_ln10/fc_lg2/fc_lge/fc_log2t/fc_log2e
fld [r1],q1 : q1 high nibble of opcode

**shift and rotate**
for RCR, RCL only last 6 bits are taken into account

**psapadd** very slow
r1=abcd
r2=efgh
new r1=[(a*d+e*h)/256] [(b*d+f*h)/256] [(c*d+g*h)/256] [d+h]

**psrpadd** very slow
r1=abcd
r2=efgh
new r1=psapadd(r1,efg[(x*(255-d)/256])

**stack operations**

| | | | |
|---|---|---|---|
| push n | cmp sp,n | add sp,n | sub sp,n |
| pop r1 | mov r1,[sp] | xchg r1,sp | xchg r1,[sp] |

EXECUTABLE HEADER
signature : MZVM
filsize : bin32;
runsize : bin32;
memsize : bin32;
entry_p : bin32;
orig_sp : bin32;
stk_siz : bin32;
endsect : bin32;

Functii sistem:

| INT 1 – Video Functions | |
|---|---|
| nr | Functia – in AX |
| 0 | Get Video Version<br>=> AX – Ver x.x.x.x in AX<br>=> BX – capabilities |
| 1 | Get Video Size<br>=> BX=horizontal size<br>=> CX=vertical size<br>=> DX=type (might be 0 if overlay not permitted)<br>**Table of video modes**<br>0 – blank screen<br>1 – textmode DOS-like<br>2 – 1 bit<br>3 – 4 bit<br>4 – 8 bit<br>5 – 16 bit (only available for overlay)<br>6 – 24 bit<br>7 – 32 bit (only available for overlay) |
| 2 | Set Video Size and Type, parameters:<br>BX=horizontal size<br>CX=vertical size<br>DX=type<br>EX=video buffer pointer<br>The memory needed for each horizontal line vill be padded with extra bytes until divisible by 4.<br>The video buffer must be large enough for the desired mode, resolution and bits per pixel.<br>For text modes AX must be divisible by 8 and BX by 16<br>Invalid calls result in video mode 0 |
| 3 | Set Video Pointer<br>EX=video buffer pointer |
| 4 | Query overlay mode<br>Same as function 1 plus EX is set to<br>0 – overlay not available<br>1 – windowed mode FX is set to needed address space<br>2 – may run in fullscreen mode |
| 5 | Initialize Overlay Video mode<br>If the application wants to run windowed type must be 0.<br>buffer must be large enough to cover the entire video adress space<br>Same as function 2 plus<br>FX – function to call when video mode changes (will be called same as an interrupt)<br>=> BX – row lenght in bytes<br>=> CX – start of window<br>=> DX – actual mode (0 if not succeeded)<br>=> EX – buffer size<br>In overlay mode the memory referenced by the video buffer pointer will be mapped over the real video memory. |
| 6 | Set Video Palette<br>BX = address<br>CX = used entries |
| 7 | Refresh entire video |
| 8 | Refresh Rectangle<br>BX = horizontal start<br>CX = vertical srart<br>DX = horizontal end<br>EX = vertical end |
| 9 | Immediately Refresh entire video |
| 10 | Immediately Refresh Rectangle same as 8 |
| 11.. | Reserved for additional Video Procedures |

| INT 2 – Messaging and I/O functions (not implemented yet) | |
|---|---|
| nr | Functia |
| 0 | Get I/O Version<br>=> AX – Ver x.x.x.x in AX<br>=> BX – capabilities |
| 1 | Wait for message parameters: AX = acceptable messages = sum of<br>1 – keyboard<br>2 – mouse<br>4 – timer<br>8 – other messages<br>Results:<br>=> AX mesasage type<br>=> R1..R7 other message parameters |
| 2 | Peek message<br>Result: like function 1 |
| 3 | Get keyboard status<br>AX pointer to 256 boolean (byte boolean) array : true if coresponding key is pressed |
| 4 | Get mouse status<br>=>AX = horizontal position<br>=>BX = vertical position<br>=>CX = button status |
| 5 | Set text cursor position<br>BX = horizontal position (upper left corner)<br>CX = vertical position |
| 6 | Show text cursor |
| 7 | Hide text cursor |
| 8 | Set text cursor size<br>BX = horizontal size (should not be larger than 255)<br>CX = vertical size -/-<br>Should not be used in overlay video modes except for mode 1 |
| 9 | Set mouse cursor position and appearence<br>BX = horizontal position (upper left corner)<br>CX = vertical position |
| 0a | Show mouse cursor |
| 0b | Hide mouse cursor |
| 0c | Set mouse cursor position and appearence<br>BX = horizontal size (0 = use curent) (divisible by 8) (should not be larger than 32)<br>CX = vertical size -/-<br>DX = pointer to shape (0 = use curent) |
| 0d.. | Reserved for additional messaging and I/O procedures |

| INT 3 – File I/O functions – directories not supported yet | |
|---|---|
| nr | Functia |
| 0 | Get File I/O Version<br>=> AX – Ver x.x.x.x in AX<br>=> BX – capabilities<br>=> CX – maximum number of simultaneous opened files |
| 1 | Create / Rewrite file<br>BX : Pchar name<br>=> CX – handle;<br>0 = could not create |
| 2 | Open file in read only mode<br>BX : Pchar name<br>=> CX – handle<br>=> DX – file length<br>0 = could not open |
| 3 | Open file in read-write mode<br>BX : Pchar name<br>=> CX – handle<br>=> DX – file length<br>0 = could not open |
| 4 | Open file in append mode – write only – cannot seek<br>BX : Pchar name<br>=> CX – handle<br>0 = could not open |
| 5 | Close file<br>CX : handle |
| 6 | Seek<br>BX : position<br>CX : handle<br>=> BX new position |
| 7 | Read<br>BX : buffer address<br>CX : handle<br>DX : number of bytes to read<br>=>DX bytes read |
| 8 | Write<br>BX : buffer address<br>CX : handle<br>DX : number of bytes to write<br>=> DX bytes written |
| 9 | Scan directory<br>BX : buffer to store names (zero-separated strings, double zero at the end)<br>CX : directory name (ignored for the moment)<br>DX : size of buffer must be at least 2<br>=> DX space used or required in buffer (0=nothing found or error) |
| | |