```
// vm1

import "io"

let twotimes(x) be
{ let sptr, prev;
  if x = 0 then resultis 0;
  prev := twotimes(x-1);
  resultis prev+2 }

/* Will use pages 6, 7, and 8 for
       the page table directory
       the page table for addresses up to 00000000001111111111111111111111
       and the page table for the stack area
   respectively. Shift page number left 11 bits to make a physical address */

let start() be
{ let pdir = 6 << 11, ptablow = 7 << 11, ptabhigh = 8 << 11;
  let a, b;
  for i = 0 to 2047 do
  { pdir ! i := 0;
    ptablow ! i := 0;
    ptabhigh ! i := 0 }

/* page table entries: bit 0 (least significant) = valid entry,
      bit 1 = only accessible in system mode,
      bits 2 - 10 not used,
      bits 11 - 31 = physical page number */

  pdir !   0 := ptablow bitor 1;
  pdir ! 511 := ptabhigh bitor 1;
  ptablow  !    0 := 1;    /* page 0 already occupied by executable code,
                              stack pointer is at 0x7FFFFFFF: just below half
                              way through the virtual address space */
  ptabhigh ! 2047 := 0x7FFFF801;
  out("pdir     = %x\n", pdir);
  out("pdir!  0 = %x\n", pdir ! 0);
  out("pdir!511 = %x\n", pdir ! 511);

  outs("Dangerous place\n");
  assembly
  { load   r1, [<pdir>]
    setsr  r1, $pdbr    // special register PDBR, page directory address
    getsr  r1, $flags   // spec reg FLAGS = all flag values in one word.
    sbit   r1, $vm
    flagsj r1, pc }     // jump to PC leaves PC unchanged.

  outs("still alive\n");
  a := 500;  // this works when a is small, but not when it is more than about 350.
             // the stack frames are 6 words long, and we have only allowed one
             // page of 2048 words for the stack.
  b := twotimes(a);
  out("twotimes(%d) = %d\n", a, b) }
```

```
// vm2: add this to the above

import "io"

manifest
{ iv_none = 0,        iv_memory = 1,      iv_pagefault = 2,   iv_unimpop = 3,
  iv_halt = 4,        iv_divzero = 5,     iv_unwrop = 6,      iv_timer = 7,
  iv_privop = 8,      iv_keybd = 9,       iv_badcall = 10,    iv_pagepriv = 11,
  iv_debug = 12,      iv_intrfault = 13 }

let ivec = table 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

let set_handler(int, fn) be
  if int >= 0 /\ int <= 13 then
    ivec ! int := fn

let int_enable() be
  assembly
  { LOAD  R1, [<ivec>]
    SETSR R1, $INTVEC
    LOAD  R1, 0
    SETFL R1, $IP }

let int_disable() be
  assembly
  { LOAD  R1, 1
    SETFL R1, $IP }

let otherhandler(intcode, address, info) be
{ out("interrupt %d (%x, %d)\n", intcode, address, info);
  ireturn }

// and in start

  set_handler(iv_pagefault, otherhandler);
  int_enable();



pdir = 3000
pdir!0 = 3801
pdir!511 = 4001
Dangerous place
still alive
INTERRUPT PAGEFAULT: Page fault (L2709)
  code=2, address=7FFFF7FF, detail=2

INTERRUPT INTRFAULT: Failure to process interrupt (L2709)
  code=13, address=7FFFF7FF, detail=2


PROCESSOR HALTED with PC = 0x0000044C
>


WHY???????

size
ssp
pc
```

```
// vm4 only showing the changes

let nextfreepage, nextentry;

let pfhandler(intcode, address, info, pc) be
{ let ptabhigh = 8 << 11;
  out("page fault for %x\n", address);
  ptabhigh ! nextentry := (nextfreepage << 11) bitor 1;
  nextentry -:=1;
  nextfreepage +:= 1;
  pc -:= 1;
  ireturn }

let start() be
{ let pdir = 6 << 11, ptablow = 7 << 11, ptabhigh = 8 << 11, ptabss = 9 << 11,
      sysstackpage = 10 << 11;
  let a, b, syssp = 0xBFFFFFFF;
  out("end is %x\n", ! 0x101);
  set_handler(iv_pagefault, pfhandler);
  int_enable();
  for i = 0 to 2047 do
  { pdir ! i := 0;
    ptablow ! i := 0;
    ptabhigh ! i := 0;
    ptabss ! i := 0; }

  pdir ! 0 := ptablow bitor 1;
  pdir ! 511 := ptabhigh bitor 1;
  pdir ! 767 := ptabss bitor 1;
  ptablow ! 0 := 1;
  ptablow ! 1 := 0x801;
  ptabhigh ! 2047 := 0x7FFFF801;
  nextentry := 2046;
  nextfreepage := 11;
  ptabss ! 2047 := sysstackpage bitor 1;

  out("pdir     = %x\n", pdir);
  out("pdir!  0 = %x\n", pdir ! 0);
  out("pdir!511 = %x\n", pdir ! 511);
  outs("Dangerous place\n");
                                                  // sys off too
  assembly
  { load   r2, sp
    load   r3, fp
    load   sp, [<syssp>]
    load   r1, [<pdir>]
    setsr  r1, $pdbr
    getsr  r1, $flags
    sbit   r1, $vm
    cbit   r1, $sys
    flagsj r1, pc
    load   sp, r2
    load   fp, r3 }

  outs("still alive\n");
  a := 500;
  b := twotimes(a);
  out("twotimes(%d) = %d\n", a, b) }
```

```
end is 9FA
pdir     = 3000
pdir!  0 = 3801
pdir!511 = 4001
Dangerous place
still alive
page fault for 7FFFF7FF
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
.....
..... blah blah blah blah
.....
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
page fault for 47FE
INTERRUPT PAGEFAULT: Page fault (L2709)
  code=2, address=BFFFF7FF, detail=0

INTERRUPT INTRFAULT: Failure to process interrupt (L2709)
  code=13, address=BFFFF7FF, detail=2

INTERRUPT INTRFAULT: Failure to process interrupt (L2709)
  code=13, address=BFFFF7FF, detail=2


PROCESSOR HALTED with PC = 0x00000439
```

physical addresses

```
// vm5

let halt_handler(intcode, address, info) be
{ out("HALT at %x\n", address);
  assembly { halt } }


let next_free_page


let non_vm_map(pdir, vpage, ppage) be
{ let ptn = vpage >> 11,
      pn = vpage bitand 0b11111111111;
  let pt = pdir ! ptn;
  out("non_vm_map(%x, %x, %x)\n", pdir, vpage, ppage);

  test ppage = -1 then
  { ppage := next_free_page << 11;
    next_free_page +:= 1 }
  else
    ppage <<:= 11;

  test (pt bitand 1) = 0 then
  {
    pt := next_free_page << 11;
    for i = 0 to 2047 do
      pt ! i := 0;
    next_free_page +:= 1;
    out("  PD[%x] := %x\n", ptn, pt bitor 1);
    pdir ! ptn := pt bitor 1;
  }

  else
    pt neqv:= 1;

  pt ! pn := ppage bitor 1;
  out("  %x[%x] := %x\n", pt, pn, ppage bitor 1) }
```

```
let vm_map(vpage, ppage) be
{ let pdir, ptn = vpage >> 11, pn = vpage bitand 0b11111111111, entry, ptpa;
  out("vm_map(%x (%x, %x), %x)\n", vpage, ptn, pn, ppage);

  if ppage = -1 then
  { ppage := next_free_page;
    next_free_page +:= 1 }

  assembly
  { getsr  r1, $PDBR
    store  r1, [<pdir>]
    add    r1, [<ptn>]
    phload r2, r1
    store  r2, [<entry>] }
  out("  PD[%x] is %x\n", ptn, entry);

  if (entry bitand 1) = 0 then
  {
    entry := (next_free_page >> 11) bitor 1;
    assembly
    { load    r1, [<pdir>]
      add     r1, [<ptn>]
      load    r2, [<entry>]
      phstore r2, r1 }
    out("  PD[%x] = %x\n", ptn, entry)
  }

  ptpa := entry neqv 1;
  entry := (ppage << 11) bitor 1;
  assembly
  { load    r1, [<ptpa>]
    add     r1, [<pn>]
    load    r2, [<entry>]
    phstore r2, r1 }
  out("  %x[%x] = %x\n", ptpa, pn, entry) }
```

```
let page_fault_handler(intcode, address, info, pc) be
{ let ptn = address >> 22, pn = (address >> 11) bitand 0b11111111111;
  vm_map(address >> 11, -1);
  pc -:= 1;
  ireturn }
```

```
let start() be
{ let pdir = 6 << 11;
  let a, b,
      syssp = 0xBFFFFFFF,
      sp = 0x7FFFFFF,
      code_page_needed = (! 0x101) >> 11;
  next_free_page := 7;

  out("end of code is at %x\n", ! 0x101);

  set_handler(iv_pagefault, page_fault_handler);
  set_handler(iv_halt, halt_handler);
  int_enable();

  for i = 0 to 2047 do
    pdir ! i := 0;
  for i = 0 to code_page_needed do
    non_vm_map(pdir, i, i);
  non_vm_map(pdir, sp >> 11, sp >> 11);
  non_vm_map(pdir, syssp >> 11, -1);

  assembly
  { load   r2, sp
    load   r3, fp
    load   sp, [<syssp>]
    load   r1, [<pdir>]
    setsr  r1, $pdbr
    getsr  r1, $flags
    sbit   r1, $vm
    cbit   r1, $sys
    flagsj r1, pc
    load   sp, r2
    load   fp, r3 }

  a := 1000;
  b := twotimes(a);
  out("twotimes(%d) = %d\n", a, b) }
```

```
$ run vm5

end of code is at A9B

non_vm_map(3000, 0, 0)
  PD[0] := 3801
  3800[0] := 1


non_vm_map(3000, 1, 1)
  3800[1] := 801


non_vm_map(3000, FFFFF, FFFFF)
  PD[1FF] := 4001
  4000[7FF] := 7FFFF801


non_vm_map(3000, 17FFFF, FFFFFFFF)
  PD[2FF] := 5001
  5000[7FF] := 4801


vm_map(FFFFE (1FF, 7FE), FFFFFFFF)
  PD[1FF] is 4001
  4000[7FE] = 5801


vm_map(FFFFD (1FF, 7FD), FFFFFFFF)
  PD[1FF] is 4001
  4000[7FD] = 6001


twotimes(1000) = 2000
HALT at 403

$
```