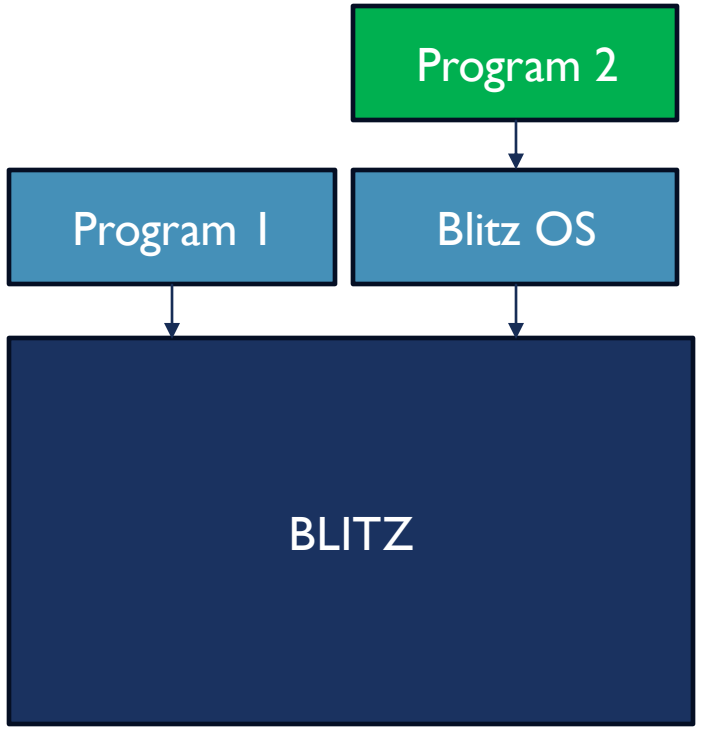
OPERATING SYSTEMS

BLITZ TOOLS

- Tools:
  - blitz
    - The emulator program. It emulates the blitz hardware.
  - asm
    - Assembler for the blitz tools. Takes in assembly, outsputs binary file.
  - kpl
    - Compiler for KPL. Takes in kpl, outputs blitz assembly.
  - lddd
    - Linker that links multiple binaries.

BLITZ EMULATOR

- Blitz is an emulator for the machine (the hardware). We're using it so we don't use our actual system.
  - Of course, as it's only a hardware, it doesn't have any OS. We have to provide it with one.
- We can still run a program directly on the machine, if configured properly.
  - For example, the programs Echo.s and Hello.s (handed out in Lab1) are meant to run directly on the machine.
- Other programs are written to run on top of an operating system that runs on Blitz.
  - The Blitz OS is so far composed of Runtime.s and System.k. It's a very basic OS at this point. We will be modifying and adding new code during our assignments.
  - An example program that run on top of the operating system: is HelloWorld.K



KPL

- Kernel Programming Language
- Very similar to C with some differences.
- We'll have examples on KPL so you can get started with it.
- If you're running into syntax issues you can:
  - Refer to documentation:
  - Ask colleagues at discord
  - Ask me.

OPERATING SYSTEM

- System.k (with System.h)
- Runtime.s

INSTALLING BLITZ

- Using in CS Labs
- Installing in your own Linux
- Using WSL

MANAGING THE STACK

Save caller function registers by pushing them to the stack.

When done, before returning, load back the saved registers from the stack using pop.

Function should always end with a return call <ret>

```
! ====================  _putString  ====================
!
! This routine is passed r1 = a pointer to a string of characters, terminated
! by '\0'.  It prints all of them except the final '\0'.  The string is printed
! atomically by calling 'debug2'.
!
! r1: ptr to string
! r2: ptr to string (saved version)
! r3: count
! r4: character
!
! Registers modified: none
!
_putString:
    push    r1          ! save registers
    push    r2          ! .
    push    r3          ! .
    push    r4          ! .
    mov r1,r2           ! r2 := ptr to the string
    mov 0,r3            ! r3 := count of characters
putStLoop:              ! loop
    loadb   [r1],r4         !   r4 := next char
    cmp r4,0            !   if (r4 == '\0')
    be  putStExit       !     then break
    add r1,1,r1         !   incr ptr
    add r3,1,r3         !   incr count
    jmp putStLoop       ! end
putStExit:              ! .
    mov 2,r1            ! perform upcall to emulator to
    debug2              ! . do the printing
    pop r4          ! restore regs
    pop r3          ! .
    pop r2          ! .
    pop r1          ! .
    ret             ! return
```

RETURN VALUE

- Result is stored on the stack at r15+4.
- r15 has the stack pointer.
- Stack pointer changes when we use push pop.
- Each push increases its value by 4.
- Careful where to store after using push.
- In GetCh, you should use storeb (for store byte) as your processing one ascii character at a time.

```
getCatchStack:
        store   r12,[r15+4]     ! put r12 in the result position
        ret                     ! return
```

ECHO.S

- This prints the input, and loops.
- You don't want that, you want to save it in the return value and end instead of looping.
- Modify the code to skip printing and return the byte read instead.
- This initializes the stack counter. DON'T do that again in Runtime.s, the stack is already initialized and you should never reset it.

```
main:
            set     STACK_START,r15 ! Initialize the stack reg
            set     SERIAL_STAT,r3  ! Initialize ptr to SERIAL_STAT word
            set     SERIAL_DATA,r4  ! Initialize ptr to SERIAL_DATA word
loop:                               ! LOOP:
wait1:                              !   WAIT1:
            load    [r3],r5         !     r5 := serial status word
            btst    0x00000001,r5   !     if status[charAvail] == 0 then
            be      wait1           !     .    goto WAIT1
            load    [r4],r2         !   Get the character
            cmp     r2,'\n'         !   if char != \n
            be      charOK          !   .
            cmp     r2,'\r'         !   .  and char != \r
            be      charOK          !   .
            cmp     r2,' '          !   .  and char < ' ' char then
            bge     charOK          !   .
            mov     '?',r2          !     char := '?'
charOK:                             !   endIf
            cmp     r2,0x7e         !   if char > 7e char then
            ble     charOK2         !   .
            mov     '?',r2          !     char := '?'
charOK2:                            !   endIf
wait2:                              !   WAIT2:
            load    [r3],r5         !     r5 := serial status word
            btst    0x00000002,r5   !     if status[outputReady] == 0 then
            be      wait2           !     .    goto WAIT2
            store   r2,[r4]         !   send char in r2 to serial output
            cmp     r2,'q'          !   if char == 'q' then
            bne     cont            !   .
            debug                   !     debug
cont:                               !   endIf
            jmp     loop            ! ENDLOOP

STACK_START     =       0x00ffff00
SERIAL_STAT     =       0x00ffff00
SERIAL_DATA     =       0x00ffff04
```

IMPORTS/EXPORTS

aFunProgram.h

```
1   header aFunProgram
2     uses System
3
4     functions
5       main ()
6
7   endHeader
```

System.h

```
-- The following routines are implemented in assembly in the Runtime.s file.
functions
  external GetCh () returns char
  external RuntimeExit ()           -- Terminate all execution and do not return
  external getCatchStack () returns int  -- Actually returns ptr to a CATCH_RECORD
  external MemoryZero (p: ptr to void, byteCount: int)  -- Set block of mem to zeros
  external MemoryCopy (destPtr: ptr to void,   -- Copy bytes from one memory area
                       srcPtr: ptr to void,    -- to another memory area.  Need not
                       byteCount: int)         -- be aligned, but must not overlap!
```

Runtime.s

```
! The following functions are implemented in this file and may be used by
! the KPL programmer.  They follow the standard KPL calling
! conventions.
!
    .export print
    .export printInt
    .export printHex
    .export printChar
    .export printBool
    .export printDouble
    .export MemoryZero
    .export MemoryCopy
    .export getCatchStack
    export GetCh
    .export RuntimeExit
```

MAKEFILE

makefile

```
HelloWorld.s: HelloWorld.h HelloWorld.k System.h
    kpl HelloWorld

HelloWorld.o: HelloWorld.s
    asm HelloWorld.s

HelloWorld: Runtime.o System.o HelloWorld.o
    lddd Runtime.o System.o HelloWorld.o -o HelloWorld
```