

The ALPACA Operating System
Draft 0.7

Scott “Jerry” Lawrence
alpaca@umlautllama.com

July 28, 2003

Contents

1	Overview	5
1.1	Goals	6
2	System Architecture	7
2.1	Hardware Architecture	7
2.2	RAM Allocation	7
2.2.1	Sprite Ram	9
2.2.2	Task Stacks	9
2.2.3	Semaphores	11
2.2.4	Message Queue	11
2.2.5	Kernel and Task Globals	11
3	System Initialization	12
3.1	Hardware Initialization	12
3.2	Display Splash Screen	14
3.3	Initialize Tasks	15
3.4	Start Runtime	16
4	Kernel Services and API	18
4.1	RST 00H - Startup/Reboot	18
4.2	RST 08H - Semaphores	19
4.3	RST 10H - Malloc	19
4.4	RST 18H - Free	19
4.5	RST 20H - Message Push/Pop	19
4.6	RST 28H - TBD	20
4.7	RST 30H - TBD	20
4.8	RST 38H - VBlank handler	20
4.9	NMI handler	20

July 28, 2003	2
5 Semaphores	21
5.1 RAM allocation	21
5.2 Locking a Semaphore	22
5.3 Releasing a Semaphore	23
6 Message Queue	24
6.1 Message Format	24
6.2 Queue Implementation	24
6.2.1 Queueing a Message	25
6.2.2 Dequeueing a Message	25
7 Memory Management	26
7.1 Memory Maintenance Structures	26
7.2 Memory Acquisition (malloc)	26
7.3 Memory Release (free)	26
8 Interrupt Service Routine	27
8.1 ISR Overall View	27
8.2 Task Switching	28
8.2.1 Design	29
8.3 Task Slot Management Initialization	32
8.3.1 Task Search / Task List	32
8.3.2 Task System Initialization	32
8.4 Task Slot Management Mechanism	34
8.4.1 Task Slot Management	35
8.4.2 Task Slot Time Management	35
9 Task Exec	36
9.1 Task Format Header	36
9.2 Task Entry Point	36
9.3 Start Task (exectask)	37
9.4 Stop Task (kill)	38

July 28, 2003	3
10 Task 0: Pac Tiny User Interface (PTUI)	39
10.1 Graphics	39
10.1.1 Cursor and Wallpaper	41
10.1.2 Flags	41
10.1.3 Frame and Dragbar	42
10.1.4 Widgets	43
10.1.5 Widget Type Flags	44
10.2 Implementation	44
10.3 Header	45
10.4 Process routine	45
11 Task 1: TBD Example	46
11.1 Header	46
11.2 Process routine	47
12 Task 2: TBD Example	48
12.1 Header	48
12.2 Process routine	49
13 Task 3: TBD Example	50
13.1 Header	50
13.2 Process routine	51
14 Utility Functions	52
14.1 memset256 - set up to 256 bytes of memory to a certian byte . .	52
14.2 memsetN - set N blocks of memory to a certian byte	53
14.3 cls - clear the screen	54
14.4 sleep - timer-based wait	55
15 System Errors	57
16 Appendix	58
A Development Schedule	59
A.1 Phase 1	59
A.2 Phase 2	59
A.3 Phase 3	59

B	Hardware memory constants	60
B.1	Pac-Man Configuration	60
B.1.1	Sprite Hardware	61
B.1.2	Sound Hardware	62
B.1.3	Enablers	62
B.1.4	Extras for Pac	62
B.2	Pengo Configuration	63
B.2.1	Sprite Hardware	64
B.2.2	Sound Hardware	64
B.2.3	Enablers	65
B.2.4	Extras for Pengo	65
C	The .asm File	66
C.1	Pac-Man ASM	66
C.2	Pengo ASM	66
C.3	Common Top	67
C.4	Common Bottom	68
D	Auxiliary Data Files	71
D.1	genroms .ROMS files	71
D.1.1	Ms. Pac-Man	71
D.1.2	Pac-Man	73
D.1.3	Pengo 2u	74
D.2	turaco .INI file	75
D.2.1	(Ms.) Pac-Man	75
D.2.2	Pengo	77
E	Building Alpaca	79
E.1	Required software	79
E.2	Makefile targets	80
E.3	The Makefile	81
F	Software License	89
F.1	The Short Version	89
F.2	The Long Version	90

Chapter 1

Overview

This document describes and implements ALPACA. ALPACA is a multitasking operating system designed for Pac-Man¹ and Pengo² arcade hardware.

This document contains all of the source code (Z-80 ASM) to build the core operating system, as well as a few example tasks. The asm file generated by this document (`alpaca.asm`) is commented as well so this document is not needed to understand what is going on in that file.³ This document can be used alone or as the reference for the generated .asm file.

Pengo is included as well for the explanations since the basic hardware is identical to Pac-Man, albiet with its control registers and layout of the hardware differing slightly. In fact, Pengo hardware is a superset of Pac-Man hardware. Anything that runs on Pac hardware should run on Pengo. Pengo adds some other hardware, like the ability to switch graphics banks, as well as some extra ram, but those details are outside of the scope of this document.

About the only main differences is that the sound and color PROMS are layed out differently. This will result in colors being "off", or the sound not sounding right.

It should also be noted that all of the graphics used in the graphics roms are all original to avoid copyright issues with either NAMCO or SEGA.

Anyway, the hardware has some distinct and extreme limitations. These limitations are:

- 1 Kb of RAM
- 16 Kb of ROM (Pac-Man hardware)

¹Pac-Man is copyright and trademark NAMCO.

²Pengo is copyright and trademark SEGA.

³I know that this goes against the reason for using noweb, but this is meant to be used as a learning device for others, and I feel that having fully documented asm is important for this purpose.

- background of 8x8 tiled characters, four colors each (1 Kb)
- 6 floating sprites (16x16 pixels, four colors) (1 Kb)

Ms. Pac-Man adds another 8Kb of non-contiguous ROM.

Pengo hardware doubles the RAM to 2 Kb, and has 36 Kb of contiguous ROM, making for a much more flexible system.

1.1 Goals

The goals of ALPACA are to provide task management, messaging, basic semaphores, simple ram management and a graphical user interface for a few tasks concurrently running on the arcade machine computer. The number of runnable tasks will be fixed. This all comes together to form a fully pre-emptive multitasking operating system can be built on such a tight hardware platform.

The design of the architecture is detailed in §2.

Chapter 2

System Architecture

This chapter explains how the kernel and memory of the system are arranged.

2.1 Hardware Architecture

First of all, we'll start with how the hardware is arranged. If you look at figure 2.1, you will see the memory map for Pac-Man based games on the left, and Pengo on the right. Pengo is only really shown as reference since it was mentioned earlier in this doc. All of the design described here will focus on Pac-Man hardware.

In a nutshell, there is some ROM on the system, shown in green. There also are some control registers which allow the program to get input from the user (joystick, coin switches, etc) which are shown in blue. This group also contains things like a flag to flip the screen, as well as the watchdog timer.

The watchdog timer is a device that resets the system completely unless it has been cleared within 16 screen refreshes. This is made for when a game might get into some unpredicted behavior where it might crash or hang. When the game gets to that state, it will reboot itself using this mechanism. We will essentially disable it by clearing it within the interrupt routine which happens once every screen refresh.

2.2 RAM Allocation

There are three groups of RAM, shown in pink in figure 2.1. These are the screen color and character RAM, as well as User RAM. The screen color and character RAM are for drawing things on the screen. The hardware has a character-based background, where you put the character to draw in the character RAM and the color to draw it in the color RAM.

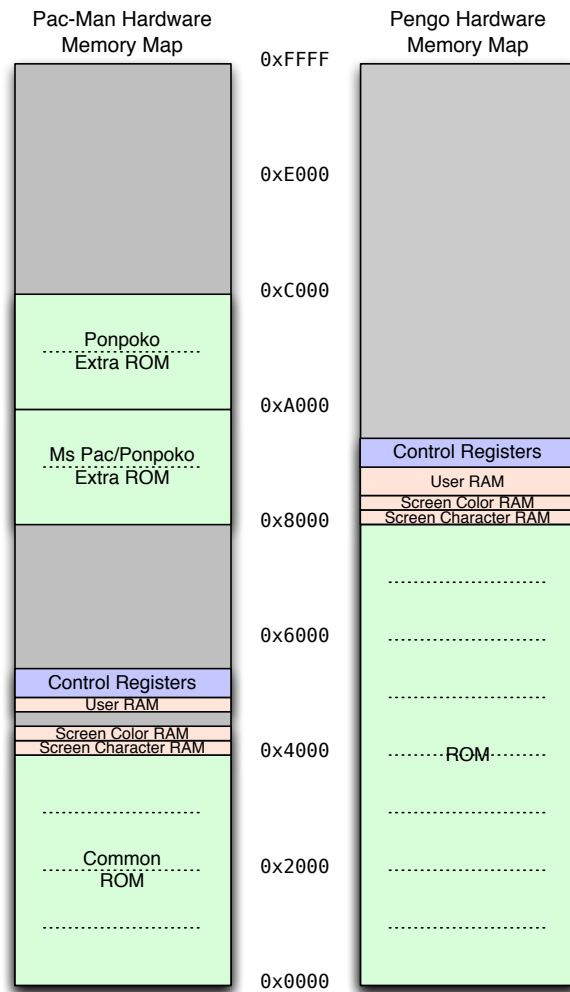


Figure 2.1: Hardware memory map

The other RAM is the User Ram, which is general purpose, for whatever the program/programmer wants to use it for. The exception is the uppermost 16 bytes, which is used to draw floating sprites on the screen.

Figure 2.2 shows just the User Ram on the system. This shows how ALPACA uses the ram. It is broken up into 6 sections. This diagram assumes that there are four tasks concurrently running. More about those in §8.

The sections shown are: (from top to bottom)

- Sprite Ram (16 bytes)
- Task 0 Stack (192 bytes)
- Task 1 Stack (192 bytes)
- Task 2 Stack (192 bytes)
- Task 3 Stack (192 bytes)
- Semaphores (16 bytes)
- Message Queue (64 bytes)
- Kernel and Task Globals (160 bytes)

2.2.1 Sprite Ram

This is a section of RAM that is used by the sprite video hardware. This is where the positions, colors, sprite numbers and flags are placed by the software to have the video hardware draw the sprites on the screen.

2.2.2 Task Stacks

Each task will have its own stack pointer and stack. Figure 2.2 shows four task stacks in the system for up to four tasks running. If we had more ram or a disk for virtual memory, we could probably increase this to be virtually unlimited, but for now, we'll stick to four.

When each task is enabled by the task switcher¹ it needs to be within its own stack frame. Each task thinks that only itself is running. There are some rudimentary communications methods by which one task can talk to another, and that is via the Message Queue, which is discussed next. Other than the Message Queue, the task has no idea if there is one other task, or thirty other tasks running on the system.

¹See §8 for more information.

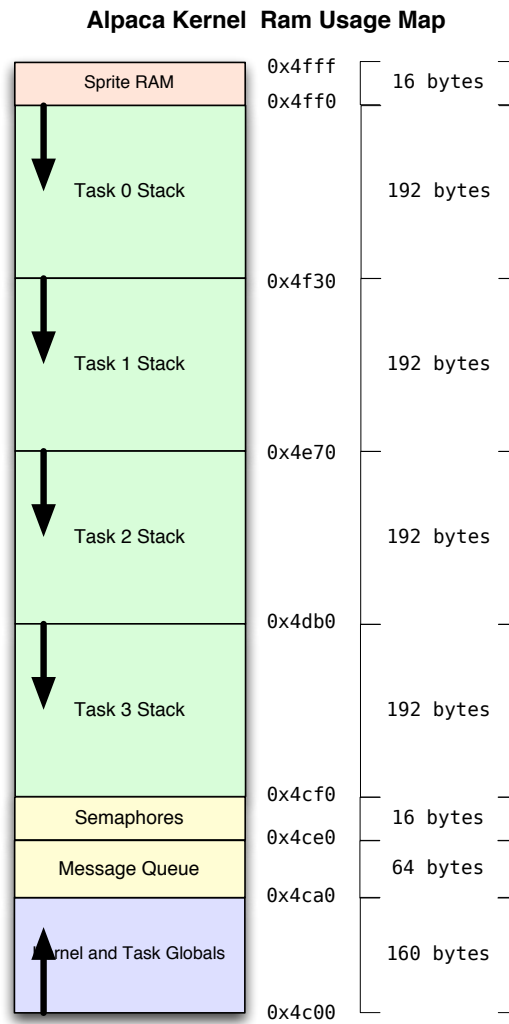


Figure 2.2: Kernel RAM memory map

2.2.3 Semaphores

This is the ram where the kernel will keep track of the state of all of the semaphores that are in use in the system. More about those in §5.

2.2.4 Message Queue

The message queue is a small amount of memory (256 bytes) that contains rudimentary messages (TBD) that allow for a task to communicate with the kernel or with other tasks.

More details about the message queue can be found in §6.

2.2.5 Kernel and Task Globals

This section of memory contains all of the variables used by the kernel itself as well as all of the tasks themselves. Since there is no memory protection at all all of this has to be coordinated such that multiple tasks are prevented from assuming control of RAM that another task or the kernel is using. Obviously, this cannot be enforced, so it is the obligation of the task to “play nice” with the other tasks, and stay within its own sandbox.

The memory allocation routines are discussed in §7.

Chapter 3

System Initialization

This chapter describes what the system does as it starts up, and how it initializes all of the hardware and software modules.

1. Hardware Initialization - zero all ram
2. Splash Screen Display
3. Initialize Tasks
4. Start Runtime

```
12a <.start implementation 12a>≡  
    .start:  
    <start hardware init 12b>  
    <start initialize tasks 15>  
    <start enable interrupts 14b>  
    <start splash screen 14d>  
    <start runtime 16>
```

This code is used in chunk 68.

3.1 Hardware Initialization

This gets called immediately from the RST 00 call, as defined in §4, which basically is simply a `jp` to here at memory location 0x0000, which is where execution starts when the processor is turned on.

Okay, so the first thing that happens is that we head over to the `.startup` block, where lots of things will be setup.

```
12b <start hardware init 12b>≡  
    di ; disable processor interrupts
```

This definition is continued in chunks 13 and 14a.

This code is used in chunk 12a.

We setup the “initial” stack pointer because this will change around once we get into starting up the multiple threads later.

```
13a <start hardware init 12b>+≡
      ld      sp, #(stack)    ; setup the initial stack pointer
```

This code is used in chunk 12a.

Interrupt mode 1 sends all interrupts through vector 0x0038, which is what we will use for the IRQ timer.

```
13b <start hardware init 12b>+≡
      im      1                ; setup interrupt mode 1
```

This code is used in chunk 12a.

For the next bit, we will use a memset function which we define in §14.

Let’s clear the watchdog timer, along with all of the other special hardware. All of the control registers are within the range of 0x5000 through 0x50c0.

```
13c <start hardware init 12b>+≡
      ;; clear the special registers
      ld      a, #0x00         ; a = 0x00
      ld      hl, #(specreg)   ; hl = start of special registers
      ld      b, #(speclen)    ; b = 0xC0 bytes to zero
      call    memset256        ; 0x5000-0x50C0 will get 0x00
```

This code is used in chunk 12a.

Now clear the sprite registers...

```
13d <start hardware init 12b>+≡
      ;; clear sprite registers
      ld      a, #0x00         ; a = 0x00
      ld      hl, #(sprtbase)  ; hl = start of sprite registers
      ld      b, #(sprtlen)    ; b = 0x10 16 bytes
      call    memset256        ; 0x4ff0-0x4fff will get 0x00
```

This code is used in chunk 12a.

Now clear the screen/video ram...

```
13e <start hardware init 12b>+≡
      ;; clear the screen ram
      call    cls              ; clear the screen RAM
```

This code is used in chunk 12a.

Next, we will need to clear the user ram. This should look very similar, since it needs to do something similar. This is a one-time use thing, so we won't bother making it a callable method. (You will never need to do this once the system is running.)

Similarly to the above, we need to clear 4 blocks of 256 bytes of ram.

```
14a  <start hardware init 12b>+≡
      ;; clear user ram
      ld    hl, #(ram)      ; hl = base of RAM
      ld    a, #0x00        ; a = 0
      ld    b, #0x02        ; b = 2 blocks of 256 bytes to clear
      call  memsetN         ; clear the blocks
```

This code is used in chunk 12a.

Once we're done with everything, we need to do some pac-specific setup for the interrupt hardware on the machine. Basically we just need to set an interrupt vector and turn on the interrupts externally.

```
14b  <start enable interrupts 14b>≡
      ;; setup pac interrupts
      ld    a, #0xff        ; fill register 'a' with 0xff
      out   (0x00), a       ; send the 0xff to port 0x00
      ld    a, #0x01        ; fill register 'a' with 0x01
```

This definition is continued in chunk 14c.

This code is used in chunk 12a.

Now we just need to enable interrupts, both in the cpu and in the external mechanism.

```
14c  <start enable interrupts 14b>+≡
      ld    (irqen), a      ; enable the external interrupt mechanism.
      ei
```

This code is used in chunk 12a.

Okay... at this point, we're ready to do something real on the machine. Everything has been set up to a state that is now known.

3.2 Display Splash Screen

We just want to display a little something while we wait for things to start up.

```
14d  <start splash screen 14d>≡
      ;; display splash screen
      nop
```

This code is used in chunk 12a.

3.3 Initialize Tasks

This is covered in /S/refsec:tasksysinit.

15 *<start initialize tasks 15>*≡
<Task System Initialization 32b>

This code is used in chunk 12a.

3.4 Start Runtime

Eventually replace this with the task executor.

```

16  <start runtime 16>≡
    ;; start runtime
    call    cls

    foo:

    ;; fill some color ram with color 1
    ld     hl, #colram
    ld     a, #0x01
    ld     b, #0x04
    call   memsetN

    ld     bc, #10
    call   sleep

    ;; fill some char ram with 'X'
    ld     hl, #vidram
    ld     a, #0x58
    ld     b, #0x04
    call   memsetN

    ld     bc, #10
    call   sleep

    jp over

    ld     a, #0x01
    call   semalock
    ld     a, #0x01
    call   semalock

over:

    ;; fill some color ram with color 8
    ld     hl, #colram
    ld     a, #0x09
    ld     b, #0x04
    call   memsetN

    ld     bc, #10
    call   sleep

    ;; fill some space with 'Y'
    ld     hl, #vidram
    ld     a, #0x59
    ld     b, #0x04
    call   memsetN

    ld     bc, #10

```

```
call    sleep

;; fill some color ram with color 0x0f
ld      hl, #colram
ld      a, #0x0F
ld      b, #0x04
call    memsetN

ld      bc, #10
call    sleep

;; fill some space with 'Z'
ld      hl, #vidram
ld      a, #0x5A
ld      b, #0x04
call    memsetN

ld      bc, #10
call    sleep

jp      foo
halt
```

This code is used in chunk 12a.

Chapter 4

Kernel Services and API

This chapter describes and defines the interface that tasks use to access the services of the OS kernel.

The services provided by the kernel are provided through the `RST` calls of the Z80 processor. There are 8 of these calls, as well as an interrupt routine that the Z80 provides. The interrupt routine is used by the task switcher, and is described in §8, however an overview of the 8 `RST` functions is provided next.

Each of these start 8 bytes off from the previous, so we need to be sure that we don't overwrite previous ones, as well as be sure that we start each of them at the right location. We can fill these with five `nops`, but instead, we'll use the `.org` directive on following calls. We just need to be sure that we don't use more than 8 bytes for each of these.

4.1 `RST 00H` - Startup/Reboot

This is the startup/reboot call. This will setup the system and restart it appropriately according to the initialization routines as defined and implemented in §3. We will just call that routine from here.

The basic initialization starts off at `0x0000` in ROM. This doubles as the implementation for `RST 00`. So we need to be sure that we are at `0x0000`. This simply jumps to the `.startup` routine.

```
18 <RST 00 implementation 18>≡
    .org 0x000
    .reset00:                ; RST 00 - Init
        jp .start
```

This code is used in chunk 68.

4.2 RST 08H - Semaphores

Semaphore control

```
19a  <RST 08 implementation 19a>≡
      .org 0x0008
      .reset08:                ; RST 08 - Semaphore control
      ret
```

This code is used in chunk 68.

4.3 RST 10H - Malloc

Allocate RAM (Malloc)

```
19b  <RST 10 implementation 19b>≡
      .org 0x0010
      .reset10:               ; RST 10 - Malloc
      ret
```

This code is used in chunk 68.

4.4 RST 18H - Free

Free RAM (Free)

```
19c  <RST 18 implementation 19c>≡
      .org 0x0018
      .reset18:              ; RST 18 - Free
      ret
```

This code is used in chunk 68.

4.5 RST 20H - Message Push/Pop

Push/ pop a message onto/off of the message stack

```
19d  <RST 20 implementation 19d>≡
      .org 0x0020
      .reset20:              ; RST 20 - Message Push / Pop
      ret
```

This code is used in chunk 68.

4.6 RST 28H - TBD

TBD

```
20a <RST 28 implementation 20a>≡
    .org 0x0028
    .reset28:                ; RST 28 - TBD
        ret
```

This code is used in chunk 68.

4.7 RST 30H - TBD

TBD

```
20b <RST 30 implementation 20b>≡
    .org 0x0030
    .reset30:                ; RST 30 - TBD
        ret
```

This code is used in chunk 68.

4.8 RST 38H - VBlank handler

VBLANK IRQ interrupt. This should never be called directly by a task. We will simply jump to the `.isr` function from here, which sits after the below NMI handler, in ROMspace.

```
20c <RST 38 implementation 20c>≡
    .org 0x0038
    .reset38:                ; RST 38 - Vblank Interrupt Service Routine
        jp     .isr
```

This code is used in chunk 68.

4.9 NMI handler

We're not using an NMI in this implementation, but we'll leave this here in case we want to use it in the future. This sits at 0x0066, 38 bytes from the RST 38 handler. We're basically wasting this space, but we might come back later and fill it in or just drop the NMI handler altogether. Regardless, this handler is here even though it's not used in Pac/Pengo hardware.

```
20d <NMI implementation 20d>≡
    .org 0x0066
    .nmi:                    ; NMI handler
        retn
```

This code is used in chunk 68.

Chapter 5

Semaphores

This chapter describes how the semaphores are managed in ALPACA.

THESE DON'T SEEM TO WORK PROPERLY YET.

NOTE: We also should disable task switching and/or interrupts when we're locking a semaphore.

5.1 RAM allocation

For now, each semaphore is a single byte. We have 16 allocated for the system, which should be more than enough for four tasks.

These are located at `semabase` in ram.

```
21  <Semaphore RAM 21>≡
      ; semaphores
      semabase      = (ram + 0x0ce0)
      semamax       = (semabase + 0x0F)
```

This code is used in chunk 68.

5.2 Locking a Semaphore

An attempt to lock a semaphore that is already locked will result in the task blocking until the semaphore is released.

We'll do some rudimentary range limiting on A by anding the passed-in semaphore number in the accumulator with 0x0F, since we only have 16 semaphores.

We then will load HL with the base address of the semaphore ram, then add in the above offset onto it.

Once it is released, it will re-set the semaphore, then return to the task.

```

22  <Semaphore lock implementation 22>≡
    ;; semalock - lock a semaphore
    ;          in    a          which semaphore to lock
    ;          out   -
    ;          mod   -
semalock:
    ; set aside registers
    push  af
    push  bc
    push  hl
    ; set up the address
    and   #0x0F          ; limit A to 0..15
    ld    c, a           ; c is the current semaphore number
    ld    b, #0x00       ; b=0 (bc = 0x00SS)
    ld    hl, (semabase) ; hl = base address
    add   hl, bc         ; hl = address of this semaphore
.s12:
    bit   1, (hl)
    jp    NZ, .s12      ; while it's set, loop
    ; set the bit
    set   1, (hl)      ; lock the semaphore
    ; restore registers
    pop   hl
    pop   bc
    pop   af
    ; return
    ret

```

This code is used in chunk 68.

5.3 Releasing a Semaphore

Releasing a semaphore is even easier than locking one.

Just like the above, we'll do some rudimentary range limiting on **A** by anding the passed-in semaphore number in the accumulator with **0x0F**, since we only have 16 semaphores.

We then will load **HL** with the base address of the semaphore ram, then add in the above offset onto it.

Then we simply clear the bit.

We can eventually combine the two of these if we want, to save a few bytes. Even easier, just after the **res** we can jump to just after the **set** in the above routine... that will save 1 or 2 bytes, but increase obfuscation quite a bit, so we won't do that just yet...

```

23  <Semaphore release implementation 23>≡
      ;; semarel - release a semaphore
      ;
      ;           in      a           which semaphore to release
      ;           out      -
      ;           mod      -
semarel:
      ; set aside registers
      push  af
      push  bc
      push  hl
      ; set up the address
      and   #0x0F           ; limit A to 0..15
      ld    c, a           ; c is the current semaphore number
      ld    b, #0x00       ; b=0 (bc = 0x000S)
      ld    hl, (semabase) ; hl = base address
      add   hl, bc         ; hl = address of this semaphore
      ; clear the semaphore
      res   1, (hl)       ; clear the bit
      ; restore registers
      pop   hl
      pop   bc
      pop   af
      ; return
      ret

```

This code is used in chunk 68.

Chapter 6

Message Queue

This chapter describes how all of the messaging in the system is handled.

6.1 Message Format

TBD

6.2 Queue Implementation

Two pointers are maintained into the Message queue; the head and tail pointers. There is also a variable which contains the number of messages currently in the queue. These variables are global for all tasks, and thus the mechanisms for queueing and dequeuing messages into the system are provided by the kernel.

```
24  <Message RAM 24>≡  
      ; messages  
      msgbase      = (ram + 0x0ca0)  
      msgmax       = (msgbase + 0x003f)
```

This code is used in chunk 68.

6.2.1 Queueing a Message

We need a way to continue adding messages onto the queue while circulating around the ram buffer, so we will have a ram buffer that is 256 bytes large, so that we can just AND the offset with 0x00FF to determine the correct offset into the message queue.

1. If number of messages is greater than 256, fail.
2. Store the message at the RAM location that the tail pointer references
3. Increment the tail pointer
4. AND the tail pointer with 0x00FF
5. Add the tail pointer with the base of the message queue
6. increment the number of messages

6.2.2 Dequeueing a Message

Similarly, we need a way to pop a message off of the queue, so a similar process is used.

1. If number of messages is 0, fail
2. Set the message at the head pointer aside
3. Increment the head pointer
4. AND the head pointer with 0x00FF
5. Add the head pointer with the base of the message queue
6. Decrement the number of messages
7. Return the message

Chapter 7

Memory Management

This chapter describes how all of the memory management (allocation and free) is performed within the system.

7.1 Memory Maintenance Structures

7.2 Memory Acquisition (malloc)

7.3 Memory Release (free)

Chapter 8

Interrupt Service Routine

This chapter describes the Interrupt Service Routine within the kernel. This chapter covers the basic Timer as well as the whole task switching routine.

8.1 ISR Overall View

Here is the overall view of the interrupt service routine, which gets called 60 times a second, when the VBLANK happens in the video hardware:

```
27a  <Interrupt Service Routine implementation 27a>≡  
      .isr:  
      <Interrupt disable interrupts and save regs 27b>  
      <Interrupt clear the watchdog 28b>  
      <Interrupt increment global timer 28d>  
      <Interrupt task switch 34>  
      <Interrupt enable interrupts and restore regs 28a>
```

This code is used in chunk 68.

We need to disable interrupts, both in the CPU as well as in the external interrupt mechanism. In the process of doing this, we will dirty up a few registers, so we might as well save them aside in here also.

```
27b  <Interrupt disable interrupts and save regs 27b>≡  
      di                ; disable interrupts (no re-entry!)  
      push    af         ; store aside some registers  
      push    bc  
      xor     a          ; a = 0  
      ld     (irqen), a  ; disable external interrupt mechanism
```

This code is used in chunk 27a.

Later on, we'll need to turn interrupts back on, and restore those registers.

```
28a  <Interrupt enable interrupts and restore regs 28a>≡
      ld    a, #0x01      ; a = 1
      ld    (irqen), a    ; enable external interrupt mechanism
      pop   bc            ; restore those regs
      pop   af
      ei                      ; enable processor interrupts
      reti                 ; return from interrupt routine
```

This code is used in chunk 27a.

Anyway, we've still got a 0 loaded into `a` from the above disabling, so we can just send that over to the watchdog as well.

Dealing with the watchdog timer in here prevents the user code (tasks) from having to deal with it at all. The original intention of the watchdog reset hardware is described in §2.1.

```
28b  <Interrupt clear the watchdog 28b>≡
      ld    (watchdog), a ; kick the dog
```

This code is used in chunk 27a.

Also, while in the interrupt routine we want to increment the global timer variable.

The timer is a value in RAM that gets updated by the IRQ/Vblank routine.

```
28c  <Timer RAM 28c>≡
      ; timer counter (word)
      timer = (ram + 17)
```

This code is used in chunk 68.

```
28d  <Interrupt increment global timer 28d>≡
      ld    bc, (timer)   ; bc = timer
      inc   bc            ; bc++
      ld    (timer), bc   ; timer = bc
```

This code is used in chunk 27a.

And that's the basics. Without the task switching, the above is a useful and fully functional ISR. The sections that follow will add in the task switching.

8.2 Task Switching

The tasks will run in the foreground, just going about their business. These tasks will be interrupted and switched out by the Task Manager from within the Interrupt routine. This will control how much time each task gets, managing their stacks, and all of that fun stuff.

The task switcher is also the backend for the `exec` and `kill` routines, which are described in §9. That is to say that when a task is instantiated with the `exec` command, or a task slot is cleared with the `kill` command, it really only sets flags directly from those commands. All of the work of setting up the task to run in a task slot is handled here in this routine.

8.2.1 Design

The design described here supports up to four concurrently running tasks, selected from up to 127 tasks available in the program ROM. There can be multiple instances of the same task running, as long as they both are careful about how they share RAM.

Each of the four tasks has its own space in RAM for their own stack. Each stack gets 0x00c0 or (192) bytes, which is hopefully enough of a stack for each task. Being that the tasks will be written in asm, this should hopefully be more than enough.

```
29a <Task Constants 29a>≡
      stacksize      = 192          ; number of bytes per stack
```

This definition is continued in chunks 30 and 31b.
This code is used in chunk 68.

And here's where we'll define the stack ram itself:

```
29b <Task Stack RAM 29b>≡
      ; stack regions for the four tasks
      stackbottom    = (stack-(stacksize*4)) ; 192 bytes (bottom of stack 3)
      stack3         = (stack-(stacksize*3)) ; 192 bytes
      stack2         = (stack-(stacksize*2)) ; 192 bytes
      stack1         = (stack-(stacksize*1)) ; 192 bytes
      stack0         = (stack)              ; top of space - sprite ram
```

This code is used in chunk 68.

This leaves 0x4c00 thru 0x4cff for program/user ram.

Task Slot Indexes

There are two bytes in RAM per slot that the kernel uses to keep track of the task running in those slots, as well as a way for the task slots to be controlled. These are the `taskslot` and `taskctrl` arrays.

The task slot indexes (`taskslot`) show which task is loaded in which task slot. This is a single byte (8 bit) index into the `tasklist`, which we will define later.

```
29c <Task RAM 29c>≡
      ; which task is in which slot (index into tasklist)
      taskslot       = (ram + 8) ; 4 bytes, one per slot
      t0slot         = (ram + 8)
      t1slot         = (ram + 9)
      t2slot         = (ram + 10)
      t3slot         = (ram + 11)
```

This definition is continued in chunks 30 and 31.
This code is used in chunk 68.

In summary:

- bit 7 == 0 : task is running
- bit 7 == 1 : special command
- bits 6..0 : currently running task ID

Special commands:

- 0xff : the slot is open for a task to be run in it.

```
30a  <Task Constants 29a>+≡
      openslot      = 0xff          ; the flag for an open slot
```

This code is used in chunk 68.

Task Slot Control

We also have the control variables which we use to set flags and values which the user functions will use to control the task switcher.

```
30b  <Task RAM 29c>+≡
      ; control information for each slot (to be handled by switcher)
      taskctrl      = (ram + 12) ; 4 bytes, one per slot
      t0ctrl        = (ram + 12)
      t1ctrl        = (ram + 13)
      t2ctrl        = (ram + 14)
      t3ctrl        = (ram + 15)
```

This code is used in chunk 68.

In summary:

- bit 7 == 0 : task in 6..0 is to be run
- bit 7 == 1 : task is a special command
- bits 6..0 : next task to run

Special commands:

- 0xff : kill the task currently running in this slot
- 0xfe : the task relinquishes its time

```
30c  <Task Constants 29a>+≡
      killslot      = 0xff          ; the flag for a slot to be killed
      freetime      = 0xfe          ; the flag for a task giving up time
```

This code is used in chunk 68.

Task Switcher Control

We also need to keep track of other things about the tasking system. This information will be stored in the `taskflag` byte in RAM.

```
31a  <Task RAM 29c>+≡
      ; various flags about the task switcher system
      taskflag      = (ram + 16)
```

This code is used in chunk 68.

One use of this byte is to determine if the task switcher should be operating. If the `taskon` bit is cleared, the task switcher will be disabled. If this bit is set, it will operate as normal.

```
31b  <Task Constants 29a>+≡
      taskon        = 7          ; multitasking is enabled (bit 7)
```

This code is used in chunk 68.

The lower three bits of the `taskflag` determine the task index number for the currently active task.

Dormant Task Stack Pointers

We need some space in RAM to hold the stack pointers for these four tasks. This is just the stack pointers for each of the running tasks. If, for example, task 1 is currently running, then the value of `t1sp` is invalid.

```
31c  <Task RAM 29c>+≡
      ; stack pointers for the four slots
      taskstack     = (ram + 0) ; 8 bytes, two per slot
      t0sp          = (ram + 0)
      t1sp          = (ram + 2)
      t2sp          = (ram + 4)
      t3sp          = (ram + 6)
```

This code is used in chunk 68.

8.3 Task Slot Management Initialization

When ALPACA starts up, it needs to set up the task slot variables to something reasonable such that we can operate normally.

8.3.1 Task Search / Task List

Future versions of the OS might include a routine that scans through ROM to find available tasks to run them. This will allow for ROMs, cartridges, or banks to be switched in while the system is live.

In the future, this will produce a 0 terminated list of pointers to the headers in RAM, but for now, we will just have this so-called `tasklist` in ROM.

This is just a list of the headers, terminated with a 0

```
32a <Task List 32a>≡
      ; list of all tasks available, null terminated
tasklist:
      .word t0header
      .word t1header
      .word t2header
      .word t3header
      .word 0x0000
```

This code is used in chunk 68.

8.3.2 Task System Initialization

Now the initialization. This sets it up such that the above ram locations have been initialized properly, and the task switcher in §8.2 knows that the task slot is empty.

First, we need clear the flags, to insure that all of the slots are open, and that the task switcher is disabled.

```
32b <Task System Initialization 32b>≡
      ;; initialize tasks
      ; clear flags
xor     a           ; a=0
ld     (taskflag), a ; clear all task flags
```

This definition is continued in chunk 33.

This code is used in chunk 15.

We initialize the stack pointers. This will get replaced in the task switcher, but for now, we will initialize it in here as well. We'll just set them all to 0x0000

```
33a  <Task System Initialization 32b>+≡
      ; clear the dormant stack pointers
      ld    bc, #0x0000    ; bc = 0x0000
      ld    (t0sp), bc    ; task 0 stack pointer gets 0x0000
      ld    (t1sp), bc    ; task 1 stack pointer gets 0x0000
      ld    (t2sp), bc    ; task 2 stack pointer gets 0x0000
      ld    (t3sp), bc    ; task 3 stack pointer gets 0x0000
```

This code is used in chunk 15.

We set all of the task slots as "open" in the slot index pointers as well. We do this by setting the indexes to the special constant, `openslot`, defined above.

```
33b  <Task System Initialization 32b>+≡
      ; all slots are open
      ld    a, #(openslot) ; a = openslot
      ld    (t0slot), a    ; set task slot 0 as open
      ld    (t1slot), a    ; set task slot 1 as open
      ld    (t2slot), a    ; set task slot 2 as open
      ld    (t3slot), a    ; set task slot 3 as open
```

This code is used in chunk 15.

Finally, enable the task switcher.

```
33c  <Task System Initialization 32b>+≡
      ; enable the task switcher
      ld    a, #(taskon)   ; a = taskon
      ld    (taskflag), a  ; set the flag
```

This code is used in chunk 15.

8.4 Task Slot Management Mechanism

This section defines the basic overall view of the Task Slot/Switching routine of the Interrupt Service Routine. The various things that can happen within this framework are defined in §8.4.1 and §8.4.2.

NOTE: This following block needs A LOT of work. I need to figure out the best way to deal with everything we know at this point, and come up with a good way to handle it quickly and efficiently.

```

34  <Interrupt task switch 34>≡
      ;; task switch stuff

      ; if switching is disabled, skip
      ; if (taskflag) & bit7 == 0
      ; jp .doneTask

      ; current task is invalid (taskslot) == 0xff
      ; skip to next task
      ; push all regs
      ; currentslot = (taskflag)
      ; inc currentslot
      ; currentslot &= 0x13
      ; (taskflag) gets currentslot
      ; jp      .skipTask

      ; current task has control MGMT items?
      ; currentslot = (taskflag) & 0x03
      ;

      ; kill current task
      ; currentslot = (taskflag) & 0x03
      ; (taskslot)+currentslot gets [openslot]

      ; start new task

      ; restart current task

      ; jp      .doneTask

      ; current task is running
      ; decrement time
      ; dec (tasktime)
      ; FALL THROUGH

      ; current task giving up its time (control)==[freetime]
      ; time set to 0
      ; ld (tasktime), 0
      ; FALL THROUGH

```

```
        ; current task ran out of time (tasktime) == 0
.skipTask:
        ; skip to next valid task
        ; inc task & 0x03 until
        ;         we're back on ourselves
        ; or
        ;         we find a valid task
        ; reset tasktime
        ; jp         .doneTask
.doneTask:
```

This code is used in chunk [27a](#).

8.4.1 Task Slot Management

Kill a task

Start a new task

Restart the current task

8.4.2 Task Slot Time Management

Let the task run

Skip remaining time

Proceed to next task

Chapter 9

Task Exec

This chapter describes how a task is started up within the ALPACA system. We also describe how a task needs to be formatted within the ROMspace such that the kernel can find the tasks, run them and interact with them.

9.1 Task Format Header

This is basically just a simple header that has all of the information that the OS needs to work with a task. The four byte cookie is there for the task searcher, which is not currently implemented, but will be in future versions of ALPACA.

- 4 bytes - magic cookie `0xc9 0x4a 0x73 0x4c` ('ret' 'J' 's' 'L') (for the searcher)
- 1 byte - task format version `0x01` (version 1)
- 1 byte - requested priority. This is the number of timeslices the task wants at a particular run between switching out.
- 2 bytes - pointer to an pascal/asciz string for task name. The data this points to should consist of a byte with the string length in it, followed immediately by that string, null terminated.
- 2 bytes - task entry point. This is just the address to the task's main routine.

9.2 Task Entry Point

This is the routine that the "exec" will jump to when the task is started up. This routine should not return. It should end with a `halt` opcode, and possibly call the `kill` routine to dequeue itself from the system, and open the slot.

9.3 Start Task (execstart)

This will take in two values. First is a value which specifies which task to run. This is used as an index into the `tasklist` array, defined in §8.3.1. Secondly, it takes in a value which specifies in which slot to run that task.

The name “execute” is really a misnomer. The task will not really be executed in this section, but rather, the task will be scheduled to be run in a specified task slot. This task will then be started within the task switcher routine, in §8.2.

And this is why all of the information about actually starting a task or killing a task (later on) is covered in §8.

In a nutshell, to start up a task in a slot, we set the task number into A, and the slot into D. This will set the control register for the specific slot at `taskctrl[d]` with the task to run. We just need to be sure that bit 7 of the task number is clear. We also need to limit the slot to [0..3].

```
37  <Exec start implementation 37>≡
    ;; execstart - starts up a new task
    ;          in      E      task number to start
    ;          in      D      task slot to use (0..3)
    ;          out     -
    ;          mod     -
execstart:
    ; save registers we're using
    push    af
    push    de
    push    bc
    push    hl
    ; limit E (task) to 127
    res    7, e      ; limit task number to 127
    ; limit D (slot)
    ld     a, d      ; a=d
    and    #0x03     ; slot is 0,1,2, or 3
    ld     c, a      ; c=a
    ld     b, #0x00  ; b=0x00, bc = 0x000S
    ; set the control value
    ld     hl, #(taskctrl) ; set up the control register
    add    hl, bc     ; hl = base + offset
    ld     (hl), e    ; taskctrl[d] = e
    ; restore the registers
    pop    hl
    pop    bc
    pop    de
    pop    af
    ; return
    ret
```

This code is used in chunk 68.

9.4 Stop Task (kill)

We also might need a way to stop or “kill” a task. In traditional *NIX systems, “kill” sends a signal to the program to tell it to stop running. We don’t have signals (yet), so we will just implement this in the same mindset as the above. We will just signal the task switcher to remove the references to this task. Again, this does not happen in here, but rather, over in §8.2.

We basically just set the value in the appropriate

```

38  <Exec kill implementation 38>≡
      ;; execkill - kills a running task
      ;          in      D      task slot to kill
      ;          out     -
      ;          mod     -
execkill:
      ; save registers we're using
      push  af
      push  de
      push  bc
      push  hl
      ; limit D (slot) and shove it into C
      ld   a, d          ; a=d
      and  #0x03        ; slot is 0,1,2, or 3
      ld   c, a          ; c=a
      ld   b, #0x00     ; b=0x00, bc = 0x000S
      ; set the control value
      ld   hl, #(taskctrl) ; set up the control register
      add  hl, bc        ; hl = base + offset
      ld   (hl), #(killslot) ; taskctrl[d] = KILL!
      ; restore the registers
      pop   hl
      pop   bc
      pop   de
      pop   af
      ; return
      ret

```

This code is used in chunk 68.

Chapter 10

Task 0: Pac Tiny User Interface (PTUI)

This chapter implements the GUI for the system called “PTUI”. This task will be loaded into the system as task number 0.

10.1 Graphics

As you can see in figures 10.1 - 10.4, The GUI widgets, window ornagements, and cursor are stored in various locations in the graphics banks. (Use the checkerboard image to identify the sprite numbers for each of the graphical elements.

The tile graphics in bank 1, figure 10.1 are pretty basic. It simply contains alphanumeric for text, as well as the widgets needed for the windows.

The sprite graphics in bank 2, figure 10.3 contain just the cursor that the joystick will be moving around for the GUI.

These banks are the same for Pac-Man and Pengo. Pengo has one other character bank, and one other sprite bank, both of which are not used for this task.



Figure 10.1: Graphics Bank 1: Tile Graphics

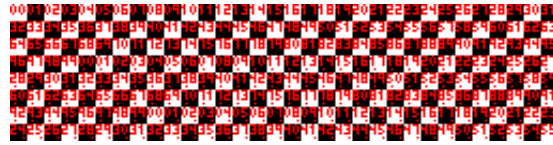


Figure 10.2: Bank 1 Checkerboard Image

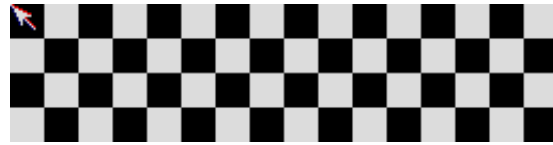


Figure 10.3: Graphics Bank 2: Sprite Graphics

This next set of blocks defines those graphical element reference numbers, as well as the colors for those elements.

```
40 <GUI Constants 40>≡
    ; GUI constants
    <GUI cursor and wallpaper 41a>
    <GUI flags 41b>
    <GUI frame and dragbar 42>
    <GUI widgets 43>
    <GUI widget types 44a>
```

This code is used in chunk 68.



Figure 10.4: Bank 2 Checkerboard Image

10.1.1 Cursor and Wallpaper

```

41a  <GUI cursor and wallpaper 41a>≡
      ; cursor and wallpaper
      PcursorS      =    1 ; sprite 1 for the cursor
      PcursorC      =    9 ; color 9 for the cursor

      PwpS          = 162 ; wallpaper sprite
      PwpC          = 0x10 ; wallpaper color 0x13- blues

      LlamaC        = 0x10 ; llama color (might be the same as PwpC above)
      CpvtC         = 0x14 ; copyright color 11

```

This code is used in chunk 40.

10.1.2 Flags

```

41b  <GUI flags 41b>≡
      ; flags
      F_Noframe     =    1 ; no frame in render (hard flag)
      F_Frame       =    2 ; frame in render (hard flag)

      F_Dirty       =    1 ; frame needs redraw (soft flag)
      F_Focus       =    2 ; frame is capturing focus currently

```

This code is used in chunk 40.

10.1.3 Frame and Dragbar

```

42  <GUI frame and dragbar 42>≡
      ; -- frame widgets --
      ; close
      PcloseS      = 128 ; close widget sprite
      PcloseCS     = 1   ; close widget selected color (5)
      PcloseCU     = 0x1e ; close widget unselected color

      ; raise
      PraiseS      = 131 ; raise widget sprite
      PraiseCS     = 1   ; raise widget selected color (5)
      PraiseCU     = 0xc  ; raise widget unselected color

      ; -- frame ornaments --
      PfrmTSEL     = 9   ; dragbar text selected color 0x14 0xb
      PfrmTUNS     = 1   ; dragbar text unselected color

      PfrmCSel    = 1   ; frame selected color
      PfrmCUns    = 0x1e ; frame unselected color

      ; bottom corners
      PSWcornS    = 138 ; southwest corner
      PSEcornS    = 139 ; southeast corner

      ; top corners
      PNWcornS    = 1   ; northwest corner 140
      PNEcornS    = 1   ; northeast corner 141

      ; top bar
      PfN_W       = 129 ; top left      (145 or 129)
      PfN_N       = 32  ; top center   (146 or 32)
      PfN_E       = 130 ; top right    (147 or 130)

      ; left bar
      PfW_N       = 132 ; left top
      PfW_W       = 133 ; left center
      PfW_S       = 134 ; left bottom

      ; right bar
      PfE_N       = 135 ; right top
      PfE_E       = 136 ; right center
      PfE_S       = 137 ; right bottom

      ; bottom bar
      PfS_W       = 142 ; bottom left
      PfS_S       = 143 ; bottom center
      PfS_E       = 144 ; bottom right

```

This code is used in chunk 40.

10.1.4 Widgets

```

43  <GUI widgets 43>≡
      ; widgets
      PwC          = 1 ; generic widget color
      PwBGS        = 127 ; window background sprite

      ; button
      PwBLuS       = 148 ; [ button left unselected sprite
      PwBRuS       = 149 ; ] button right unselected sprite

      ; selected button
      PwBLsS       = 150 ; [[ button left selected sprite
      PwBRsS       = 151 ; ]] button right selected sprite

      ; checkbox
      PwcuS        = 152 ; [ ] checkbox unselected sprite
      PwcsS        = 153 ; [X] checkbox selected sprite

      ; radio box
      PwruS        = 154 ; ( ) radio unselected sprite
      PwrsS        = 155 ; (X) radio selected sprite

      ; slider
      PwsnS        = 156 ; === slider notch sprite
      PwsbS        = 157 ; |= slider bar sprite

      ; progress bar
      PwpoS        = 158 ; progress bar open sprite
      PwpfS        = 159 ; ### progress bar filled sprite

      ; spin
      PwHsS        = 160 ; <> horizontal spin controller
      PwVsS        = 161 ; ^v vertical spin controller

```

This code is used in chunk 40.

10.1.5 Widget Type Flags

```

44a  <GUI widget types 44a>≡
      ; Widget Types (for the frame-widget table)

      W_End          = 0 ; end of the widget list
      W_Frame       = 1 ; window frame (needs to be first)

      ; frame flags:
      FF_Border     = 1 ; use a border on the frame
      FF_NClose    = 2 ; no close button
      FF_NRaise    = 4 ; no raise button

      W_MButton     = 2 ; momentary button
      W_SButton     = 3 ; sticky button

      W_Radio       = 4 ; radio button (flags is the group number)
      W_Check       = 5 ; check button

      W_SText       = 6 ; static text (text is the idx of a string)
      W_DText       = 7 ; dynamic text (data is idx of ram)

      W_DInt        = 8 ; dynamic integer (data is idx in the ram)

      W_HSlider     = 9 ; horizontal slider
      W_VSlider     = 10 ; vertical slider

      W_HSpin       = 11 ; horizontal spin
      W_VSpin       = 12 ; vertical spin

```

This code is used in chunk 40.

10.2 Implementation

```

44b  <Task 0 implementation 44b>≡
      ;; Task 0 - PTUI
      <Task 0 header 45a>

      <Task 0 process routine 45b>

```

This code is used in chunk 68.

10.3 Header

```

45a  <Task 0 header 45a>≡
      t0header:
          .byte  0xc9, 0x4a, 0x73, 0x4c  ; cookie
          .byte  0x01                    ; version
          .byte  0x04                    ; requested timeslices
          .word  t0name                  ; name
          .word  t0process               ; process function

      t0name:
          .byte  6                      ; strlen
          .asciz "Task 0"              ; name

```

This code is used in chunk 44b.

10.4 Process routine

```

45b  <Task 0 process routine 45b>≡
      t0process:
          ld     hl, #(colram)          ; base of color ram
          ld     a, #0x01               ; clear the screen to 0x00
          ld     b, #0x04               ; 256*4 = 1k
          call  memsetN                ; do it.

      t0p2:
          ld     hl, #(vidram)         ; base of video ram
          ld     a, #0x41               ; 'A'
          ld     b, #0x04               ; 256*4 = 1k
          call  memsetN

          ld     hl, #(vidram)         ; base of video ram
          ld     a, #0x42               ; 'B'
          ld     b, #0x04               ; 256*4 = 1k
          call  memsetN

          ld     hl, #(vidram)         ; base of video ram
          ld     a, #0x43               ; 'C'
          ld     b, #0x04               ; 256*4 = 1k
          call  memsetN

          jp     t0p2
          halt

```

This code is used in chunk 44b.

Chapter 11

Task 1: TBD Example

This chapter implements a simple task which will be loaded into the system as task number 1.

```
46a  <Task 1 implementation 46a>≡  
      ;; Task 1 - TBD  
      <Task 1 header 46b>
```

```
      <Task 1 process routine 47>
```

This code is used in chunk 68.

11.1 Header

```
46b  <Task 1 header 46b>≡  
      t1header:  
          .byte  0xc9, 0x4a, 0x73, 0x4c  ; cookie  
          .byte  0x01                      ; version  
          .byte  0x04                      ; requested timeslices  
          .word  t1name                    ; name  
          .word  t1process                 ; process function  
  
      t1name:  
          .byte  6                          ; strlen  
          .asciz "Task 1"                  ; name
```

This code is used in chunk 46a.

11.2 Process routine

```
47  <Task 1 process routine 47>≡
    t1process:
        ld    hl, #(colram)    ; base of color ram
        ld    a, #0x01        ; clear the screen to blue
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        ld    hl, #(colram)    ; base of color ram
        ld    a, #0x09        ; clear the screen to red
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        jp    t1process
    halt
```

This code is used in chunk 46a.

Chapter 12

Task 2: TBD Example

This chapter implements a simple task which will be loaded into the system as task number 2.

```
48a  <Task 2 implementation 48a>≡  
      ;; Task 2 - TBD  
      <Task 2 header 48b>
```

```
      <Task 2 process routine 49>
```

This code is used in chunk 68.

12.1 Header

```
48b  <Task 2 header 48b>≡  
      t2header:  
          .byte  0xc9, 0x4a, 0x73, 0x4c  ; cookie  
          .byte  0x01                      ; version  
          .byte  0x04                      ; requested timeslices  
          .word  t2name                    ; name  
          .word  t2process                 ; process function  
  
      t2name:  
          .byte  6                          ; strlen  
          .asciz "Task 2"                  ; name
```

This code is used in chunk 48a.

12.2 Process routine

```

49  <Task 2 process routine 49>≡
    t2process:
        ld    hl, #(colram)    ; base of color ram
        ld    a, #0x01        ; clear the screen to 0x00
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        ld    hl, #(vidram)   ; base of video ram
        ld    a, #0x61        ; 'a'
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        ld    hl, #(vidram)   ; base of video ram
        ld    a, #0x62        ; 'b'
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        ld    hl, #(vidram)   ; base of video ram
        ld    a, #0x63        ; 'c'
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        jp    t2process
    halt

```

This code is used in chunk 48a.

Chapter 13

Task 3: TBD Example

This chapter implements a simple task which will be loaded into the system as task number 3.

```
50a  <Task 3 implementation 50a>≡  
      ;; Task 3 - TBD  
      <Task 3 header 50b>
```

```
      <Task 3 process routine 51>
```

This code is used in chunk 68.

13.1 Header

```
50b  <Task 3 header 50b>≡  
      t3header:  
          .byte  0xc9, 0x4a, 0x73, 0x4c  ; cookie  
          .byte  0x01                      ; version  
          .byte  0x04                      ; requested timeslices  
          .word  t3name                    ; name  
          .word  t3process                 ; process function  
  
      t3name:  
          .byte  6                          ; strlen  
          .asciz "Task 3"                  ; name
```

This code is used in chunk 50a.

13.2 Process routine

```

51  <Task 3 process routine 51>≡
    t3process:
        ld    hl, #(colram)    ; base of color ram
        ld    a, #0x01        ; clear the screen to 0x00
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        ld    hl, #(vidram)    ; base of video ram
        ld    a, #0x78        ; 'X'
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        ld    hl, #(vidram)    ; base of video ram
        ld    a, #0x79        ; 'Y'
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        ld    hl, #(vidram)    ; base of video ram
        ld    a, #0x7a        ; 'Z'
        ld    b, #0x04        ; 256*4 = 1k
        call  memsetN

        jp    t3process
    halt

```

This code is used in chunk 50a.

Chapter 14

Utility Functions

This chapter describes and implements a few functions that are usable by tasks, and have some sort of utility value.

14.1 `memset256` - set up to 256 bytes of memory to a certian byte

Here we will implement a function that sets a region of memory to a certian value. Load the value into `a`, the base address into `hl`, and the number of bytes into `b`. We might want to use this in task space, so we'll make it a utility function.

```
52  <Utils memset256 implementation 52>≡
      ;; memset256 - set up to 256 bytes of ram to a certain value
      ;          in   a       value to poke
      ;          in   b       number of bytes to set 0x00 for 256
      ;          in   hl      base address of the memory location
      ;          out   -
      ;          mod   hl, bc
memset256:
      ld      (hl), a      ; *hl = 0
      inc    hl           ; hl++
      djnz   memset256    ; decrement b, jump to memset256 if b>0
      ret
      ; return
```

This code is used in chunk 68.

14.2 memsetN - set N blocks of memory to a certian byte

Here we will implement a function that sets a region of memory to a certian value. Load the value into `a`, the base address into `hl`, and the number of blocks of 256 bytes into `b`. We might want to use this in task space, so we'll make it a utility function.

```
53  <Utils memsetN implementation 53>≡
      ;; memsetN - set N blocks of ram to a certain value
      ;          in    a      value to poke
      ;          in    b      number of blocks to set
      ;          in    hl     base address of the memory location
      ;          out   -
      ;          mod   hl, bc
memsetN:
      push    bc          ; set aside bc
      ld     b, #0x00     ; b = 256
      call   memset256   ; set 256 bytes
      pop    bc          ; restore the outer bc
      djnz  memsetN      ; if we're not done, set another chunk.
      ret                    ; otherwise return
```

This code is used in chunk 68.

14.3 cls - clear the screen

The screen ram is two chunks of ram from 0x4000 through 0x43FF as well as 0x4400 through 0x47FF.

We'll basically nest two loops, both using the `djnz`. The inner loop happens in the `memset` function. The outer loop happens 8 times, since we need to do 256 bytes 8 times. (`djnz` only looks at 8 bits of register 'b'.)

```
54  <Utils cls implementation 54>≡
      ;; cls - clear the screen (color and video ram)
      ;          in      -
      ;          out     -
      ;          mod     -
cls:
      push    hl          ; set aside some registers
      push    af
      push    bc

      ld     hl, #(vidram) ; base of video ram
      ld     a, #0x05      ; clear the screen to 0x00
      ld     b, #0x08      ; need to set 256 bytes 8 times.

      call   memsetN      ; do it.

      pop    bc          ; restore the registers
      pop    af
      pop    hl
      ret                    ; return
```

This code is used in chunk 68.

14.4 sleep - timer-based wait

One thing that is very useful to have is a way for a process to wait for a specified amount of time. This is accomplished through this “sleep” command. The task puts the number of ticks to wait (60 per second) into BC then calls this routine.

Future versions might relinquish remaining clock cycles to other tasks by this communicating somehow to the task switcher, but this one just sits in a loop, waiting for the clock to be the right value.

But for this version, we will compute the timeout `current time + ticks to wait`, and just store it in BC while we loop.

The loop simply loads the current time into HL, then subtracts BC from it. We then compare it with a `sbc`, and loop if we’re not there yet.

NOTE that this is not completely accurate. There might be 1-N more ticks between when this routine returns past when you expect it to return. This is due to the multitasking nature of /OS. Your timer might be up, but another task has the processing cycles currently. As soon as we have the cpu again, we will time out and return.

```
55  <Utils sleep implementation 55>≡
      ;; sleep - wait a specified number of ticks
      ;          in      bc      number of ticks to wait
      ;          out     -
      ;          mod     -
sleep:
      ; set side some registers
      push  bc
      push  af
      push  hl
      ; compute the timeout into BC
      ld   hl, (timer)    ; hl = timer
      add  hl, bc         ; hl += ticks to wait
      push hl             ; bc =
      pop  bc             ;   = hl
      .slp:
      ; loop until the timeout comes
      ld   hl, (timer)    ; hl = current time
      sbc  hl, bc         ; set flags
      jp   M, .slp        ; if (HL >= BC) then JP .slp2
      ; restore the registers
      pop  hl
      pop  af
      pop  bc
      ; return
      ret
```

This code is used in chunk 68.

Here's what I had originally wrote. Notice that it keeps the timeout persistent by keeping it on the stack. This required an extra pop and push for each iteration through the loop, and also required an extra push and pop wrapped around that.

The above implementation only uses the stack to move the value of hl over into bc, and that happens once per call.

```
56  <original sleep implementation 56>≡
      ;; oldsleeep - wait a specified number of ticks
      ;           in      bc      number of ticks to wait
      ;           out     -
      ;           mod     -
oldsleeep:
      ; set aside some registers
      push  bc
      push  af
      push  hl
      ; compute the timeout into HL
      ld   hl, (timer)    ; hl = timer
      add  hl, bc         ; hl += ticks to wait
      push hl             ; top of stack now contains the timeout value
.slp2:
      ; loop until the timeout comes
      pop  hl             ; restore hl...
      push hl            ; ...and shove it back on the stack
      ld   bc, (timer)   ; bc = current time
      sbc  hl, bc        ; set flags
      jp   P, .slp2     ; if (HL < BC) then JP .slp2
      pop  hl
      ; restore the registers
      pop  hl
      pop  af
      pop  bc
      ; return
      ret
```

Root chunk (not used in this document).

Chapter 15

System Errors

This chapter describes how system errors are handled in ALPACA.

The System error routines are formatted similarly to the task routines. When the kernel finds an error during its interrupt routine, it will push the correct address for the error routine then return from the interrupt handler.

Each error routine should disable interrupts, clear the watchdog timer, and draw some kind of informative information on the screen for the user to see.

Errors are currently unimplemented.

Chapter 16

Appendix

Appendix A

Development Schedule

The development cycles for ALPACA have been broken down into a few phases. Each of the phases will be completed before then next one will be started.

A.1 Phase 1

- task startup with hardcoded entry points
- task switching with hardcoded priorities/delays
- compile-time memory definitions
- init and process routines for tasks

A.2 Phase 2

- task exec with ROM Task searcher
- simple message queue (not useful)

A.3 Phase 3

- task switching with wait(0), requested priorities
- more advanced message queue
- shutdown routine for tasks
- Malloc and Free implemented
- perhaps allow for multiple execs of the same process (this collides with the searcher's functionality)

Appendix B

Hardware memory constants

This chapter lists off all of the addresses for all of the bits of hardware that we will have to deal with. This chapter includes information about Pac-Man as well as Pengo hardware.

B.1 Pac-Man Configuration

60a `<PAC Global Constants 60a>≡`
`stack = 0x4ff0`

This definition is continued in chunks 60–62.
This code is used in chunk 66a.

60b `<PAC Global Constants 60a>+≡`
`vidram = 0x4000`
`colram = 0x4400`
`ram = 0x4c00`
`dsw0 = 0x5080`
`in1 = 0x5040`
`in0 = 0x5000`
`specreg = 0x5000`
`speclen = 0x00C0`
`sprtbase = 0x4ff0`
`sprtlen = 0x0010`

This code is used in chunk 66a.

And the bits for IN0:

```
61a  <PAC Global Constants 60a>+≡
      p1_up      = 0x01
      p1_left    = 0x02
      p1_right   = 0x04
      p1_down    = 0x08
      racktest   = 0x10
      coin1      = 0x20
      coin2      = 0x40
      coin3      = 0x80
```

This code is used in chunk 66a.

As well as IN1:

```
61b  <PAC Global Constants 60a>+≡
      p2_up      = 0x01
      p2_left    = 0x02
      p2_right   = 0x04
      p2_down    = 0x08
      service    = 0x10
      start1     = 0x20
      start2     = 0x40
      cabinet    = 0x80
```

This code is used in chunk 66a.

B.1.1 Sprite Hardware

This constants 8 pairs of two bytes:

- byte 1, bit 0 - Y flip
- byte 1, bit 1 - X flip
- byte 1, bits 2-7 - sprite image number
- byte 2 - color

```
61c  <PAC Global Constants 60a>+≡
      spritebase = 0x4ff0
```

This code is used in chunk 66a.

And there are 8 sprites total:

```
61d  <PAC Global Constants 60a>+≡
      nsprites   = 0x08
```

This code is used in chunk 66a.

And for the coordinates, these are xy pairs for 8 sprites.

```
61e  <PAC Global Constants 60a>+≡
      spritecoords = 0x5060
```

This code is used in chunk 66a.

B.1.2 Sound Hardware

Three voices. Voice 1:

```
62a  <PAC Global Constants 60a>+≡
      v1_acc      = 0x5040
      v1_wave     = 0x5045
      v1_freq     = 0x5050
      v1_vol      = 0x5055
```

This code is used in chunk 66a.

Voice 2:

```
62b  <PAC Global Constants 60a>+≡
      v2_acc      = 0x5046
      v2_wave     = 0x504a
      v2_freq     = 0x5056
      v2_vol      = 0x505a
```

This code is used in chunk 66a.

Voice 3:

```
62c  <PAC Global Constants 60a>+≡
      v3_acc      = 0x504b
      v3_wave     = 0x504f
      v3_freq     = 0x505b
      v3_vol      = 0x505f
```

This code is used in chunk 66a.

B.1.3 Enablers

```
62d  <PAC Global Constants 60a>+≡
      irqen      = 0x5000
      sounden    = 0x5001
      flipscreen = 0x5003
      coincount  = 0x5007
      watchdog   = 0x50C0
```

This code is used in chunk 66a.

B.1.4 Extras for Pac

```
62e  <Pac Global Constants 62e>≡
      strt1mp1   = 0x5004
      strt1mp2   = 0x5005
      coinlock   = 0x5006
```

Root chunk (not used in this document).

B.2 Pengo Configuration

63a *<PENGO Global Constants 63a>*≡
 stack = 0x8ff0
 This definition is continued in chunks 63–65.
 This code is used in chunk 66b.

63b *<PENGO Global Constants 63a>*+≡
 vidram = 0x8000
 colram = 0x8400
 ram = 0x8800
 dsw0 = 0x9040
 in1 = 0x9080
 in0 = 0x90c0
 specreg = 0x9000
 speclen = 0x00ff
 sprtbase = 0x8ff2
 sprtlen = 0x0010

This code is used in chunk 66b.

And the bits for IN0:

63c *<PENGO Global Constants 63a>*+≡
 p1_up = 0x01
 p1_down = 0x02
 p1_left = 0x04
 p1_right = 0x08
 coin1 = 0x10
 coin2 = 0x20
 coin3 = 0x40
 button1 = 0x80

This code is used in chunk 66b.

As well as IN1:

63d *<PENGO Global Constants 63a>*+≡
 p2_up = 0x01
 p2_down = 0x02
 p2_left = 0x04
 p2_right = 0x08
 service = 0x10
 start1 = 0x20
 start2 = 0x40
 button1 = 0x80

This code is used in chunk 66b.

B.2.1 Sprite Hardware

This constants 8 pairs of two bytes:

- byte 1, bit 0 - Y flip
- byte 1, bit 1 - X flip
- byte 1, bits 2-7 - sprite image number
- byte 2 - color

64a \langle PENGO Global Constants 63a $\rangle + \equiv$
 spritebase = 0x8ff2

This code is used in chunk 66b.

And there are 8 sprites total:

64b \langle PENGO Global Constants 63a $\rangle + \equiv$
 nsprites = 0x06

This code is used in chunk 66b.

And for the coordinates, these are xy pairs for 8 sprites.

64c \langle PENGO Global Constants 63a $\rangle + \equiv$
 spritecoords = 0x9022

This code is used in chunk 66b.

B.2.2 Sound Hardware

Three voices. Voice 1:

64d \langle PENGO Global Constants 63a $\rangle + \equiv$
 v1_wave = 0x9005
 v1_freq = 0x9011
 v1_vol = 0x9015

This code is used in chunk 66b.

Voice 2:

64e \langle PENGO Global Constants 63a $\rangle + \equiv$
 v2_wave = 0x900a
 v2_freq = 0x9016
 v2_vol = 0x901a

This code is used in chunk 66b.

Voice 3:

64f \langle PENGO Global Constants 63a $\rangle + \equiv$
 v3_wave = 0x900f
 v3_freq = 0x901b
 v3_vol = 0x901f

This code is used in chunk 66b.

B.2.3 Enablers

65a *⟨PENGO Global Constants 63a⟩*+≡
 irqen = 0x9040
 sounden = 0x9041
 flipscreen = 0x9043
 coincount = 0x9044
 watchdog = 0x9070

This code is used in chunk 66b.

B.2.4 Extras for Pengo

65b *⟨PENGO Global Constants 63a⟩*+≡
 palbank = 0x9042
 collutbank = 0x9046
 spritebank = 0x9047

This code is used in chunk 66b.

Appendix C

The .asm File

This is where we gather together all of the asm blocks defined above into two cohesive .asm files.

C.1 Pac-Man ASM

```
66a <pacalpaca.asm 66a>≡
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; PacAlpaca.asm
    ;
    ; ALPACA: A Multitasking operating system for Pac-Man Z80 arcade hardware
    ;
    <commontop.asm 67>
    <PAC Global Constants 60a>
    <commonbottom.asm 68>
```

Root chunk (not used in this document).

C.2 Pengo ASM

```
66b <pengoalpaca.asm 66b>≡
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; PengoAlpaca.asm
    ;
    ; ALPACA: A Multitasking operating system for Pengo Z80 arcade hardware
    ;
    <commontop.asm 67>
    <PENGO Global Constants 63a>
    <commonbottom.asm 68>
```

Root chunk (not used in this document).

C.3 Common Top

```

67  <common_top.asm 67>≡
    ; Written by
    ;     Scott "Jerry" Lawrence
    ;     alpaca@umlautllama.com
    ;
    ; This source file is covered by the LGPL:
    ;
    <license short version 89>
    ;

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;
;           This file is machine generated.  Do not edit it by hand!
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

        .title alpaca
        .module alpaca

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; some constants:

```

This code is used in chunk 66.

C.4 Common Bottom

```

68  <commonbottom.asm 68>≡

    ; constants for the task system
    <Task Constants 29a>

    ; constants for the GUI system:
    <GUI Constants 40>

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; RAM allocation:
    <Task RAM 29c>
    <Timer RAM 28c>
    <Message RAM 24>
    <Semaphore RAM 21>
    <Task Stack RAM 29b>

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; area configuration
    ; we want absolute dataspace, with this area called "CODE"
    .area    .CODE (ABS)

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; RST functions

    ; RST 00
    <RST 00 implementation 18>

    ; RST 08
    <RST 08 implementation 19a>

    ; RST 10
    <RST 10 implementation 19b>

    ; RST 18
    <RST 18 implementation 19c>

    ; RST 20
    <RST 20 implementation 19d>

    ; RST 28
    <RST 28 implementation 20a>

    ; RST 30
    <RST 30 implementation 20b>

    ; RST 38

```

```

<RST 38 implementation 20c>

; NMI
<NMI implementation 20d>

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; interrupt service routine:
<Interrupt Service Routine implementation 27a>

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; the core OS stuff:
<.start implementation 12a>

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; some helpful utility functions

; memset256
<Utils memset256 implementation 52>

; memsetN
<Utils memsetN implementation 53>

; clear screen
<Utils cls implementation 54>

; sleep
<Utils sleep implementation 55>

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; semaphore control

; lock semaphore
<Semaphore lock implementation 22>

; release semaphore
<Semaphore release implementation 23>

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; task exec and kill routines

<Exec start implementation 37>

```

(Exec kill implementation 38)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;  
; The tasks  
  
; task list -- list of all available tasks  
(Task List 32a)
```

```
;;;;;;;;;;;;;;;;;  
; task number 0  
(Task 0 implementation 44b)
```

```
;;;;;;;;;;;;;;;;  
; task number 1  
(Task 1 implementation 46a)
```

```
;;;;;;;;;;;;;;;;  
; task number 2  
(Task 2 implementation 48a)
```

```
;;;;;;;;;;;;;;;;  
; task number 3  
(Task 3 implementation 50a)
```

This code is used in chunk 66.

Appendix D

Auxiliary Data Files

This chapter defines all of the extra files needed to convert the generated ASM as well as the auxiliary PCX image files into the ROM files that we need to generate.

The two types of files, `.ROMS` and `.INI` are needed for the external `genroms` and `turacoCL` programs, which are used to generate the ROM images.

D.1 `genroms` `.ROMS` files

These files are the data files used by “`genroms`” to produce ROM image files from the generated Intel Hex File (`.IHX`) by the makefile.

The basic fields are:

- start address
- rom size
- rom filename
- rom reference name

D.1.1 Ms. Pac-Man

```
71 <mspacman.roms 71>≡
# program space
begin program
0x0000 0x1000 boot1 program_1
0x1000 0x1000 boot2 program_2
0x2000 0x1000 boot3 program_3
0x3000 0x1000 boot4 program_4
0x8000 0x1000 boot5 program_5
```

```
0x9000 0x1000 boot6  program_6
end

# graphics bank 1
begin graphics
0x0000 0x1000 5e      graphics_1

# graphics bank 2
0x0000 0x1000 5f      graphics_2
end

# color proms
begin color
0x0000 0x0020 82s123.7f      palette
0x0020 0x0100 82s126.4a      colorlookup
end

# sound proms
begin sound
0x0000 0x0100 82s126.1m      sound_a
0x0100 0x0100 82s126.3m      sound_timing
end
```

Root chunk (not used in this document).

D.1.2 Pac-Man

```

73  <pacman.roms 73>≡
    # program space
    begin program
    0x0000 0x1000 pacman.6e      program_1
    0x1000 0x1000 pacman.6f      program_2
    0x2000 0x1000 pacman.6h      program_3
    0x3000 0x1000 pacman.6j      program_4
    end

    # graphics bank 1
    begin graphics
    0x0000 0x1000 pacman.5e      graphics_1

    # graphics bank 2
    0x0000 0x1000 pacman.5f      graphics_2
    end

    # color proms
    begin color
    0x0000 0x0020 82s123.7f      palette
    0x0020 0x0100 82s126.4a      colorlookup
    end

    # sound proms
    begin sound
    0x0000 0x0100 82s126.1m      sound_a
    0x0100 0x0100 82s126.3m      sound_timing
    end

```

Root chunk (not used in this document).

D.1.3 Pengo 2u

```

74  <pengo2u.roms 74>≡
    begin program
    0x0000 0x1000 pengo.u8          program_1
    0x1000 0x1000 pengo.u7          program_2
    0x2000 0x1000 pengo.u15         program_3
    0x3000 0x1000 pengo.u14         program_4
    0x4000 0x1000 pengo.u21         program_5
    0x5000 0x1000 pengo.u20         program_6
    0x6000 0x1000 pengo.u32         program_7
    0x7000 0x1000 pengo.u31         program_8
    end

    # graphics bank 1
    begin graphics
    0x0000 0x2000 ic92              graphics_1

    # graphics bank 2
    0x0000 0x2000 ic105             graphics_2
    end

    # color and palette proms proms
    begin color
    0x0000 0x0020 pr1633.078        palette
    0x0020 0x0400 pr1634.088        colorlookup
    end

    # sound proms
    begin sound
    0x0000 0x0100 pr1635.051        sound_a
    0x0100 0x0100 pr1636.070        sound_timing
    end

```

Root chunk (not used in this document).

D.2 turaco .INI file

These files are used to convert the .pcx files into graphics ROM image files by “turacoCL”. The exact format of this file will not be described here since it is outside of the scope of this document.

For more detail about what is going on here, please refer to the documentation and sample .ini driver contained in the “turacoCL” package.

D.2.1 (Ms.) Pac-Man

```
75 <pacman.ini 75>≡
  [Turaco]
  FileVersion = 1.0
  DumpVersion = 2
  Author = Jerry / MAME 0.65.1 Dump
  URL = http://www.cis.rit.edu/~jerry/Software/turacoCL

  [General]
  Name = pacman
  Grouping = pacman
  Year = 1980
  Manufacturer = [Namco] (Midway license)
  CloneOf = puckman
  Description = Pac-Man (Midway)

  [Layout]
  GfxDecodes = 2

  [GraphicsRoms]
  Rom1 =      0      4096  pacman.5e
  Rom2 =  4096      4096  pacman.5f

  [Decode1]
  start = 0
  width = 8
  height = 8
  total = 256
  orientation = 0
  planes = 2
  planeoffsets = 0 4
  xoffsets = 56 48 40 32 24 16 8 0
  yoffsets = 64 65 66 67 0 1 2 3
  charincrement = 128

  [Decode2]
  start = 4096
  width = 16
  height = 16
```

```

total = 64
planes = 2
planeoffsets = 0 4
xoffsets = 312 304 296 288 280 272 264 256 56 48 40 32 24 16 8 0
yoffsets = 64 65 66 67 128 129 130 131 192 193 194 195 0 1 2 3
charincrement = 512

```

```

[Palette]
Palette1 = 4   0  0  0   220 220 220   0  0  90  220  0  0
Palette2 = 4   0  0  0     0 220  0   0  0  90  220 150 20
Palette3 = 4   0  0  0     0  0 220  255  0  0  255 255  0
Palette4 = 4   0  0  0   220  0  0   90 90  0  220 220 220
Palette5 = 4   0  0  0   220  0  0   0 220  0  220 220 220
Palette6 = 4   0  0  0  150 150  0   0 220  0   90 90  0
Palette7 = 4   0  0  0   220 220  0   90 90 220  220 220 220
Palette8 = 4   0  0  0   220  0  0   90 90  0  220 220 220
Palette9 = 4   0  0  0     0 150 220   0 220  0  220 220 220
Palette10 = 4  0  0  0     0  0  0   90 90 220  220 220 220
Palette11 = 4 255  0  0  255 255 255   0 255  0   0  0 220
Palette12 = 4   0  0  0  255 255 255   0  0  0   0  0 220

```

Root chunk (not used in this document).

D.2.2 Pengo

```
77 <pengo2u.ini 77>≡
  [General]
  Description = Pengo (set 2 not encrypted)

  [Layout]
  GfxDecodes = 4
  Orientation = 5

  [GraphicsRoms]
  Rom1 = 0 8192 ic92
  Rom2 = 8192 8192 ic105

  [Decode1]
  start = 0
  width = 8
  height = 8
  total = 256
  planes = 2
  planeoffsets = 0 4
  xoffsets = 64 65 66 67 0 1 2 3
  yoffsets = 0 8 16 24 32 40 48 56
  charincrement = 128

  [Decode2]
  start = 4096
  width = 16
  height = 16
  total = 64
  planes = 2
  planeoffsets = 0 4
  xoffsets = 64 65 66 67 128 129 130 131 192 193 194 195 0 1 2 3
  yoffsets = 0 8 16 24 32 40 48 56 256 264 272 280 288 296 304 312
  charincrement = 512

  [Decode3]
  start = 8192
  width = 8
  height = 8
  total = 256
  planes = 2
  planeoffsets = 0 4
  xoffsets = 64 65 66 67 0 1 2 3
  yoffsets = 0 8 16 24 32 40 48 56
  charincrement = 128

  [Decode4]
  start = 12288
  width = 16
```

```
height = 16
total = 64
planes = 2
planeoffsets = 0 4
xoffsets = 64 65 66 67 128 129 130 131 192 193 194 195 0 1 2 3
yoffsets = 0 8 16 24 32 40 48 56 256 264 272 280 288 296 304 312
charincrement = 512
```

```
[Palette]
Palette1 = 4  0  0  0  220 220 220  0  0  90  220  0  0
Palette2 = 4  0  0  0  0 220  0  0  0  90  220 150 20
Palette3 = 4  0  0  0  0  0 220 255 0  0 255 255  0
```

Root chunk (not used in this document).

Appendix E

Building Alpaca

This chapter explains what is necessary to build ALPACA, as well as how to do so.

E.1 Required software

To start off with, you will need some software packages installed to build anything:

To do anything:

- gnu make (gmake)
- noweb/notangle
- unix tools: cat, cd, cp, dd, uname, zip

To build the document:

- ImageMagick tools: convert
- LaTeX / PDFLaTeX

To build the romset:

- genroms
- turaco CL
- ZCC package or asz80 and aslink

To test the romset:

- MAME or some other emulator

E.2 Makefile targets

Once you have the correct software installed, as explained in the previous section, you should just be able to type “`gmake`”¹ and have it build this document `docs/alpaca-development.pdf` as well as the rom image files as specified in the makefile. See below on how to specify Pac-Man or Pengo roms.

As a side effect, a well commented Z80 ASM file will be in “`code/alpaca.asm`” for your viewing pleasure. To make things a little easier to see, you might want to do a `make listing` to generate the “`code/alpaca.lst`” listing file.

In a nutshell, you can just type `make targetname` to make that specific target’s files. The valid targets are:

paclisting builds: `code/pacalpaca.lst` listing file

pacprog builds: `code/pacalpaca.asm`, `code/pacfinal.ihx`

pacroms builds: `roms/pacman/*` (graphics and code)

pacromzip builds a zip of the above roms

pactest builds the above roms, runs MAME to test them out

pengolisting builds: `code/pengoalpaca.lst` listing file

pengoprogram builds: `code/pengoalpaca.asm`, `code/pengofinal.ihx`

pengoroms builds: `roms/pengo2u/*` (graphics and code)

pengoromzip builds a zip of the above roms

pengotest builds the above roms, runs MAME to test them out

docs builds: `doc/alpaca.pdf`

dview builds: `doc/alpaca.pdf`, runs `acroread`

clean gets rid of all targets

tidy cleans the doc directory of intermediate files

all builds: `doc/alpaca.pdf`, `code/pacalpaca.asm`, `code/pacalpaca.lst`, `code/pengoalpaca.asm`, `code/pengoalpaca.lst`, `pac` and `pengo` rom image files into `roms/`

dist builds: “all”, then puts it in a new directory

You may need to change the paths to the MAME program and ROM directories in the makefile if you want to run the test targets on your system.

¹or “make” on OS X

E.3 The Makefile

```

81 <GNUmakefile 81>≡
# GNUMakefile for the Alpaca project
#
#   Scott "Jerry" Lawrence
#
# It's not pretty.  Sorry about that.
#
#
# $Id: build.nw,v 1.8 2003/07/28 18:10:48 jerry Exp $
#
#####
# Targets:
#   paclisting      builds: code/pacalpaca.lst listing file
#   pacprog         builds: code/pacalpaca.asm, code/pacfinal.ihx
#   pacroms         builds: roms/pacman/pacman.* (graphics and code)
#   pacromzip       builds a zip of the above roms
#   pactest         builds the pac-man roms, runs MAME to test them out
#
#   pengolisting   builds: code/pacalpaca.lst listing file
#   pengoprogram   builds: code/pacalpaca.asm, code/pacfinal.ihx
#   pengoroms      builds: roms/pengo/pengo.* (graphics and code)
#   pengoromzip    builds a zip of the above roms
#   pengotest      builds the pengo roms, runs MAME to test them out
#
#   docs           builds: doc/alpaca.pdf
#   dview          builds: doc/alpaca.pdf, runs acroread
#
#   clean          gets rid of all targets
#   tidy           cleans the doc directory of intermediate files
#
#   dist           web-ready distribution
#
#   all            builds: docs, roms, listing
#####
all: docs paclisting pengolisting pacroms pengoroms
#####
test: pactest
#####

HAS_NOWEB := 1

#####

# program name
PROG      := alpaca

```

```
VERSION := 0.6

# directories
CODEDIR := code
ROMSROOT := roms
ROMSOURCE := roms/dummy
DISTDIR := $(PROG)_$(VERSION)

# extra programs
GENROMS := genroms
TURACOCL := turacocl
DD := dd
ZIP := zip
BLDSYS := $(shell uname -s)

ARCH := PACMAN

ifdef HAS_NOWEB

NWS := \
    nws/title.nw \
    nws/overview.nw \
    nws/arch.nw \
    nws/init.nw \
    nws/kernserv.nw \
    nws/semaphores.nw \
    nws/messages.nw \
    nws/malloc.nw \
    nws/isr.nw \
    nws/exec.nw \
    nws/task0.nw \
    nws/task1.nw \
    nws/task2.nw \
    nws/task3.nw \
    nws/utills.nw \
    nws/error.nw \
    \
    nws/appendix.nw \
    nws/schedule.nw \
    nws/hardware.nw \
    nws/asm.nw \
    nws/auxdata.nw \
    nws/build.nw \
    nws/license.nw \
    nws/end.nw

PCX :=\
    gfx/pac_1.pcx \
```

```

        gfx/pac_1c.pcx \
        gfx/pac_2.pcx \
        gfx/pac_2c.pcx

PCXPDF := $(PCX:%.pcx=%.pdf)

endif

STYLE := doc/alpaca.sty
DOC := doc/$(PROG).pdf

docs:   $(DOC)

dview:  docs
        open $(DOC)

#####

PACTARG := $(CODEDIR)/pacfinal.ihx
PACASMS := $(CODEDIR)/pacalpaca.asm

PENGOTARG := $(CODEDIR)/pengofinal.ihx
PENGOASMS := $(CODEDIR)/pengoalpaca.asm

DEPS :=

DATA :=

CLEAN := Release Build $(DISTDIR)

ifdef HAS_NOWEB
    CLEAN += $(PENGOTARG) $(PENGOTARG:%.ihx=%.map)
    CLEAN += $(PENGOASMS) $(PENGOASMS:%.asm=%.rel)
    CLEAN += $(PACTARG) $(PACTARG:%.ihx=%.map)
    CLEAN += $(PACASMS) $(PACASMS:%.asm=%.rel)
    CLEAN += doc/alpaca* code/*.lst
    CLEAN += roms/pacman/pacman.* pac*.zip
    CLEAN += roms/pengo2u/pengo*. * pengo*.zip
    CLEAN += roms/pengo2u/ic*
endif

TIDY := $(COMMON_OBJS) $(STYLE) \
        $(DOC:%.pdf=%.tex) $(DOC:%.pdf=%.aux) \
        $(DOC:%.pdf=%.log) $(DOC:%.pdf=%.toc) \
        $(PCXPDF) $(DOC:%.pdf=%.out)

#####
# Pac builds

```

```
# there should be one of these for pac, one for mspac and one for pengo
PACROMDIR      := $(ROMSROOT)/pacman
PACBACKDIR     := ../..
PACGENROMSFILE := $(CODEDIR)/pacman.roms
PACTURACOINI   := $(CODEDIR)/pacman.ini
PACROMNAME     := pacman
```

```
CLEAN += $(PACGENROMSFILE)
CLEAN += $(PACTURACOINI)
```

```
pacprog:      $(PACTARG)
.PHONY: pacprog
```

```
pacroms:      $(PACTARG) $(PACGENROMSFILE) $(PACTURACOINI)
              cd $(PACROMDIR) ; $(GENROMS) $(PACBACKDIR)/$(PACGENROMSFILE) $(PACBACKDIR)/$(PACTARG)
              $(DD) if=/dev/zero of=$(PACROMDIR)/pacman.5e bs=4096 count=1
              $(DD) if=/dev/zero of=$(PACROMDIR)/pacman.5f bs=4096 count=1
              $(TURACOCL) -inf IMG -bnk 1 -rod $(PACROMDIR) -rom $(PACROMDIR) -ini $(PACTURACOINI) -dbf gt
              $(TURACOCL) -inf IMG -bnk 2 -rod $(PACROMDIR) -rom $(PACROMDIR) -ini $(PACTURACOINI) -dbf gt
.PHONY: pacroms
```

```
pacromzip:    pacroms
              mkdir $(PACROMNAME)
              cp $(PACROMDIR)/8* $(PACROMDIR)/p* $(PACROMNAME)
              $(ZIP) -r $(PACROMNAME).zip $(PACROMNAME)
              rm -rf $(PACROMNAME)
.PHONY: pacromzip
```

```
#####
```

```
# PAC test targets
```

```
MAME := xmamed
MAMED := xmamed --debug
```

```
# pacman info
```

```
PMTRD := /Applications/jerry/Games/MacPacMAME\ 0.58/ROMS/pengman
PMTAPP := /Applications/jerry/Games/MacPacMAME\ 0.58/MacPacMAME\ 0.58
```

```
# automagically choose the correct one..
```

```
ifeq ($(BLDSYS), Darwin)
pactest:      pacroms osxpactest
else
pactest:      pacroms mamepactest
endif
.PHONY: pactest
```

```
osxpactest:
```

```

        cp -f $(PACROMDIR)/pacman.* $(PMTRD)
        cp -f $(PACROMDIR)/82*.* $(PMTRD)
        open -a $(PMTAPP)
.PHONY: osxpactest

mamepactest:
        $(MAMED) -rp $(ROMSROOT) pacman
.PHONY: mamepactest

#####
# Pengo builds

PENGOROMDIR      := $(ROMSROOT)/pengo2u
PENGOBACKDIR     := ../..
PENGOGENROMSFILE := $(CODEDIR)/pengo2u.roms
PENGOTURACOINI   := $(CODEDIR)/pengo2u.ini
PENGOROMNAME     := pengo2u

CLEAN += $(PENGOGENROMSFILE)
CLEAN += $(PENGOTURACOINI)

pengoprogram:    $(PENGOTARG)
.PHONY: pengoprogram

pengoroms:      $(PENGOTARG) $(PENGOGENROMSFILE) $(PENGOTURACOINI)
        cd $(PENGOROMDIR) ; $(GENROMS) $(PENGOBACKDIR)/$(PENGOGENROMSFILE) $(PENGOBACKDIR)/$(PENGOTARG)
        $(DD) if=/dev/zero of=$(PENGOROMDIR)/ic92 bs=8192 count=1
        $(DD) if=/dev/zero of=$(PENGOROMDIR)/ic105 bs=8192 count=1
        $(TURACOCL) -inf IMG -bnk 1 -rod $(PENGOROMDIR) -rom $(PENGOROMDIR) -ini $(PENGOTURACOINI) -
        $(TURACOCL) -inf IMG -bnk 2 -rod $(PENGOROMDIR) -rom $(PENGOROMDIR) -ini $(PENGOTURACOINI) -
        $(TURACOCL) -inf IMG -bnk 3 -rod $(PENGOROMDIR) -rom $(PENGOROMDIR) -ini $(PENGOTURACOINI) -
        $(TURACOCL) -inf IMG -bnk 4 -rod $(PENGOROMDIR) -rom $(PENGOROMDIR) -ini $(PENGOTURACOINI) -
.PHONY: pengoroms

pengoromzip:    pengoroms
        mkdir $(PENGOROMNAME)
        cp $(PENGOROMDIR)/ic* $(PENGOROMDIR)/p* $(PENGOROMNAME)
        $(ZIP) -r $(PENGOROMNAME).zip $(PENGOROMNAME)
        rm -rf $(PENGOROMNAME)
.PHONY: pengoromzip

#####
# PENGO test targets

# pengo info

```

```

PGTRD := /Applications/jerry/Games/MacMAME/ROMs/pengo2u
PGTAPP := /Applications/jerry/Games/MacMAME/MacMAME.app

# automagically choose the correct one..
ifeq ($(BLDSYS),Darwin)
pengotest:      pengoroms osxpengotest
else
pengotest:      pengoroms mamepengotest
endif
.PHONY: pengotest

osxpengotest:
cp -f $(PENGOROMDIR)/pengo.* $(PGTRD)
cp -f $(PENGOROMDIR)/ic* $(PGTRD)
cp -f $(PENGOROMDIR)/pr163*.* $(PGTRD)
open -a $(PGTAPP)
.PHONY: osxpengotest

mamepengotest:
$(MAMED) -rp $(ROMSROOT) pengo2u
.PHONY: mamepengotest

#####

clean: tidy
rm -rf $(CLEAN)

tidy:
rm -rf $(TIDY)

dist: docs paclisting pacromzip pengolisting pengoromzip
rm -rf $(DISTDIR)
mkdir $(DISTDIR)
cp $(DOC) $(PACLSTS) $(PACASMS) $(PACROMNAME).zip $(DISTDIR)
cp $(PENGOLSTS) $(PENGOASMS) $(PENGOROMNAME).zip $(DISTDIR)

#####

PACRELS      := $(PACASMS:%.asm=%.rel)
PACLSTS      := $(PACASMS:%.asm=%.lst)

PENGORELS    := $(PENGOASMS:%.asm=%.rel)
PENGOLSTS    := $(PENGOASMS:%.asm=%.lst)

paclisting:  $(PACLSTS)
pengolisting: $(PENGOLSTS)

```

```

%.lst: %.asm
      asz80 -l $<
.SECONDARY: $(PACASMS) $(PENGOASMS)

OPTS      := -O

$(PACTARG): $(PACRELS)
      aslink -i -m -o $(PACTARG) -b_CODE=0x0000 $(PACRELS)

$(PENGOTARG): $(PENGOARELS)
      aslink -i -m -o $(PENGOTARG) -b_CODE=0x0000 $(PENGOARELS)

%.rel: %.asm
      asz80 $<

%.rel: %.c
      zcc -c -v $(OPTS) -D$(ARCH) -D$(TEST) -I../include $(ADDS) $<

.SECONDARY: $(PACTARG)
.SECONDARY: $(PENGOTARG)

#####

ifdef HAS_NOWEB

$(CODEDIR)/%.asm:      $(NWS)
      -@$(MKDIR_CMD)
      notangle -R$*.asm $^ | cpif $@

$(CODEDIR)/%.roms:    $(NWS)
      -@$(MKDIR_CMD)
      notangle -R$*.roms $^ | cpif $@

$(CODEDIR)/%.ini:     $(NWS)
      -@$(MKDIR_CMD)
      notangle -R$*.ini $^ | cpif $@

%.pdf: %.tex
      -@$(MKDIR_CMD)
      ( \
        cd $(@D); \
        oldFingerprint="ZZZ" ; \
        if [ -f $*.aux ]; then \
          fingerprint="'sum $*.aux'" ; \
        else \
          fingerprint="YYY" ; \
        fi ; \
        while [ ! "$${oldFingerprint}" = "$${fingerprint}" ]; do \

```

```

        oldFingerprint="$$fingerprint" ; \
        pdflatex $(<F) ; \
        fingerprint="'sum $(*F).aux'" ; \
    done ; \
)

$(DOC:%.pdf=%.tex):    $(PCXPDF) $(NWS)
    -@$(MKDIR_CMD)
    cat $(NWS) | noweave -delay -index | cpif $@

doc/%.sty: nws/%.sty
    -@$(MKDIR_CMD)
    cp $< $@

%.pdf: %.pcx
    convert $< $@

endif

#####

.PHONY: all
.PHONY: docs
.PHONY: clean
.PHONY: tidy

#.SECONDARY: $(TIDY)

#####

$(DOC): $(PCXPDF) $(STYLE)

#####
Root chunk (not used in this document).

```

Appendix F

Software License

This software, “Alpaca” is covered by the GNU Lesser General Public License. The terms of this license are covered as follows:

F.1 The Short Version

```
89  <license short version 89>≡
    ;; Alpaca - A Multitasking operating system for Z80 arcade hardware
    ;; Copyright (C) 2003 Scott "Jerry" Lawrence
    ;; alpaca@umlautllama.com
    ;;
    ;; This is free software; you can redistribute it and/or modify
    ;; it under the terms of the GNU Lesser General Public License
    ;; as published by the Free Software Foundation; either version
    ;; 2 of the License, or (at your option) any later version.
    ;;
    ;; This software is distributed in the hope that it will be
    ;; useful, but WITHOUT ANY WARRANTY; without even the implied
    ;; warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
    ;; PURPOSE. See the GNU Lesser General Public License for
    ;; more details.
    ;;
    ;; You should have received a copy of the GNU Lesser General
    ;; Public License along with this library; if not, write to
    ;; the Free Foundation, Inc., 59 Temple Place, Suite 330,
    ;; Boston, MA 02111-1307 USA
```

This code is used in chunk 67.

F.2 The Long Version

90 *<license long version 90>*≡

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it. You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use,
not price. Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights. These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you. You must make sure that they, too, receive or can get the source
code. If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free

programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does

and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the

entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it

contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

- b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that

system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN

WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Root chunk (not used in this document).