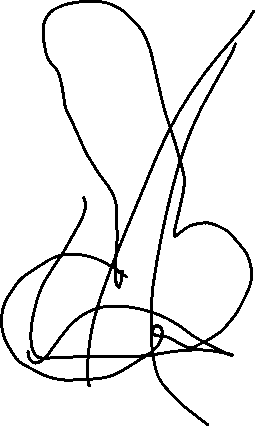
# Instructions

## Instruction Word Format



All instructions for BARNDLES are assembled into fixed-length 24-bit instructions. The most significant byte of each instruction is composed of an op code, an immediate flag I, a CPSR update flag S, and an invert flag N. The following bytes vary depending on the type of the instruction. Non-immediate instructions are composed of 4-bits of reserved space, a first operand register Rm, a second operand register Rn, and a destination register Rd. Immediate instructions are composed of a 12-bit immediate value Imm12 and a destination register Rd.



Figure .: The BARNDLES non-immediate instruction format



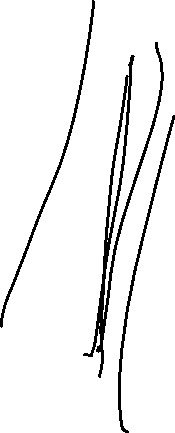
Figure .: The BARNDLES immediate instruction format

* + - 1. Op Code

The op code specifies which operation the microprocessor should execute on its operands.

* + - 1. Immediate Flag (I)

The immediate flag specifies whether the instruction should be decoded as an immediate instruction or as a non-immediate instruction. If the immediate flag is set, the instruction will be decoded as an immediate instruction. Otherwise, it will be decoded as a non-immediate instruction.



* + - 1. CPSR Update Flag (S)

The CPSR update flag S specifies whether or not the current program status register CPSR should be updated to store the ALU status flags set by the execution of the instruction. If the CPSR update flag is set, the CPSR will be updated on writeback. Otherwise, the CPSR will not be updated.

* + - 1. Invert Flag (N)

The invert flag N specifies whether or not the value output by the ALU after the execution of the instruction should be inverted before writeback. If the invert flag is set, the ALU output will be inverted. Otherwise, it will not be affected.

* + - 1. First Operand Register (Rm)

The first operand register is the register that contains the first operand for the operation. This field is ignored if the instruction is in the immediate format (the immediate flag I is set) or if the instruction does not require a first operand register to be specified.

* + - 1. Second Operand Register (Rn)

The second operand register is the register that contains the second operand for the operation. This field is ignored if the instruction is in the immediate format (the immediate flag I is set) or if the instruction does not require a second operand register to be specified.

* + - 1. Destination Register (Rd)

The destination register is the register that the output from the operation will be stored in. This field is ignored if the instruction does not require a destination register to be specified.

* + - 1. 12-bit Immediate (Imm12)

The 12-bit immediate field holds a 12-bit immediate value that can be used as an operand in immediate instructions. This field is ignored if the instruction is in the non-immediate format (the immediate flag I is cleared) or if the instruction does not require a 12-bit immediate to be specified.

## Assembly Instruction Format

### Optional Suffixes

Optional suffixes can be optionally appended to the end of an instruction to add additional functionality to it. Any number of optional suffixes can be appended to a single instruction. The order that the optional suffixes are appended in does not matter to the assembler, but convention suggests that suffixes be appended in alphabetical order. The optional suffixes provided by BARNDLES ASM are as follows.

* + - 1. Invert Flag (N)

The invert flag will cause the output of the ALU to be inverted before writeback. This is useful for creating additional operations from existing functions. For example, you can convert the MOV instruction to a move not instruction MOVN or the AND instruction to a NAND instruction ANDN.

* + - 1. CPSR Update Flag (S)

The CPSR update flag will update the current program status register CPSR to reflect the ALU status flags produced by the instruction that the update flag is appended to. This allows the programmer to optimize their code by partially eliminating the need for compare instructions CPS and CPA.

### Flexible Address Operand

Flexible address operands allow the programmer to use one of multiple formats for specifying a memory address. Upon assembly, the flexible operand is replaced with a 12-bit immediate containing the memory address that the programmer’s flexible operand references. Given they are converted to immediates, flexible address operands should only be used in immediate instructions. The flexible address operands provided by BARNDLES ASM are as follows.

* + - 1. Label (label)

The label operand allows you to reference the memory location of a label in the assembly program.

* + - 1. 12-bit Immediate (#Imm12)

The immediate operand allows you to directly specify a memory location by its 12-bit address.

## Instruction Set

### Do Nothing (DON)

The DON instruction does nothing.

* + - 1. Non-Immediate Syntax

DON

* + - 1. Non-Immediate Operation

None

* + - 1. Immediate Syntax

DON

* + - 1. Immediate Operation

None

### Move (MOV)

The MOV instruction moves an integer into a register or between registers.

* + - 1. Non-Immediate Syntax

MOV{N,S} Rd, Rm

* + - 1. Non-Immediate Operation

Rd <= Rm

* + - 1. Immediate Syntax

MOV{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Add (ADD)

The ADD instruction adds one integer to another integer and stores the sum in a register.

* + - 1. Non-Immediate Syntax

ADD{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm + Rn

* + - 1. Immediate Syntax

ADD{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd + Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Subtract (SUB)

The SUB instruction subtracts one integer from another integer and stores the difference in a register.

* + - 1. Non-Immediate Syntax

SUB{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm - Rn

* + - 1. Immediate Syntax

SUB{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd - Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Multiply (MUL)

The MUL instruction multiplies one integer by another integer and stores the product in a register.

* + - 1. Non-Immediate Syntax

MUL{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm × Rn

* + - 1. Immediate Syntax

MUL{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd × Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Divide (DIV)

The DIV instruction divides one integer by another integer and stores the product in a register.

* + - 1. Non-Immediate Syntax

DIV{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm ÷ Rn

* + - 1. Immediate Syntax

DIV{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd ÷ Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Bitwise AND (AND)

The AND instruction bitwise ANDs one integer by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

AND{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm & Rn

* + - 1. Immediate Syntax

AND{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd & Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Bitwise OR (ORR)

The ORR instruction bitwise ORs one integer by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

ORR{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm | Rn

* + - 1. Immediate Syntax

ORR{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd | Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Bitwise XOR (XOR)

The ORR instruction bitwise XORs one integer by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

XOR{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm ^ Rn

* + - 1. Immediate Syntax

XOR{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd ^ Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Logical AND (LGA)

The LGA instruction logically ANDs one integer by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

LGA{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm && Rn

* + - 1. Immediate Syntax

LGA{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd && Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Logical OR (LGO)

The LGO instruction logically ORs one integer by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

LGO{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm || Rn

* + - 1. Immediate Syntax

LGO{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd || Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Logical XOR (LGX)

The LGX instruction logically XORs one integer by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

LGX{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm ^^ Rn

* + - 1. Immediate Syntax

LGX{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd ^^ Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Logical Shift Left (LSL)

The LSL instruction logically shifts an integer left by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

LSL{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm LSL Rn

* + - 1. Immediate Syntax

LSL{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd LSL Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Logical Shift Right (LSR)

The LSR instruction logically shifts an integer right by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

LSR{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm LSR Rn

* + - 1. Immediate Syntax

LSR{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd LSR Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Arithmetic Shift Right (ASR)

The ASR instruction arithmetically shifts an integer right by another integer and stores the result in a register.

* + - 1. Non-Immediate Syntax

ASR{N,S} Rd, Rm, Rn

* + - 1. Non-Immediate Operation

Rd <= Rm ASR Rn

* + - 1. Immediate Syntax

ASR{N,S} Rd, #Imm12

* + - 1. Immediate Operation

Rd <= Rd ASR Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Branch (BCH)

The BCH instruction assigns a new value to the program counter register PC.

* + - 1. Non-Immediate Syntax

BCH Rd

* + - 1. Non-Immediate Operation

PC <= Rd

* + - 1. Immediate Syntax

BCH <AddrOp>

* + - 1. Immediate Operation

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.

### Branch and Link (BAL)

The BAL instruction stores the current value of the program counter register PC in the link register LR and assigns a new value to the program counter register.

* + - 1. Non-Immediate Syntax

BAL Rd

* + - 1. Non-Immediate Operation

LR <= PC

PC <= Rd

* + - 1. Immediate Syntax

BAL <AddrOp>

* + - 1. Immediate Operation

LR <= PC

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.

### Branch If Equal (BEQ)

The BEQ instruction assigns a new value to the program counter register PC if the last instruction to update the current program status register CPSR produced a zero result.

* + - 1. Non-Immediate Syntax

BEQ Rd

* + - 1. Non-Immediate Operation

if CPSR[Z]:

PC <= Rd

* + - 1. Immediate Syntax

BEQ <AddrOp>

* + - 1. Immediate Operation

if CPSR[Z]:

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.
* Z is the ALU’s zero status flag as read from the current program status register CPSR.

### Branch If Not Equal (BNE)

The BNE instruction assigns a new value to the program counter register PC if the last instruction to update the current program status register CPSR produced a non-zero result.

* + - 1. Non-Immediate Syntax

BNE Rd

* + - 1. Non-Immediate Operation

if not CPSR[Z]:

PC <= Rd

* + - 1. Immediate Syntax

BNE <AddrOp>

* + - 1. Immediate Operation

if not CPSR[Z]:

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.
* Z is the ALU’s zero status flag as read from the current program status register CPSR.

### Branch If Greater Than (BGT)

The BGT instruction assigns a new value to the program counter register PC if the last instruction to update the current program status register CPSR produced a non-negative and non-zero result.

* + - 1. Non-Immediate Syntax

BGT Rd

* + - 1. Non-Immediate Operation

if not CPSR[N] and not CPSR[Z]:

PC <= Rd

* + - 1. Immediate Syntax

BGT <AddrOp>

* + - 1. Immediate Operation

if not CPSR[N] and not CSPR[Z]:

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.
* Z is the ALU’s zero status flag as read from the current program status register CPSR.
* N is the ALU’s negative status flag as read from the current program status register CPSR.

### Branch If Less Than (BLT)

The BLT instruction assigns a new value to the program counter register PC if the last instruction to update the current program status register CPSR produced a negative result.

* + - 1. Non-Immediate Syntax

BLT Rd

* + - 1. Non-Immediate Operation

if CPSR[N]:

PC <= Rd

* + - 1. Immediate Syntax

BLT <AddrOp>

* + - 1. Immediate Operation

if CPSR[N]:

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.
* N is the ALU’s negative status flag as read from the current program status register CPSR.

### Branch If Greater Than or Equal To (BGE)

The BGE instruction assigns a new value to the program counter register PC if the last instruction to update the current program status register CPSR produced a non-negative result.

* + - 1. Non-Immediate Syntax

BGE Rd

* + - 1. Non-Immediate Operation

if not CPSR[N]:

PC <= Rd

* + - 1. Immediate Syntax

BGE <AddrOp>

* + - 1. Immediate Operation

if not CPSR[N]:

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.
* N is the ALU’s negative status flag as read from the current program status register CPSR.

### Branch If Less Than or Equal To (BLE)

The BLE instruction assigns a new value to the program counter register PC if the last instruction to update the current program status register CPSR produced a zero or negative result.

* + - 1. Non-Immediate Syntax

BLE Rd

* + - 1. Non-Immediate Operation

if CPSR[N] or CPSR[Z]:

PC <= Rd

* + - 1. Immediate Syntax

BLE <AddrOp>

* + - 1. Immediate Operation

if CPSR[N] or CPSR[Z]:

PC <= <AddrOp>

Where:

* Rd is the destination register.
* <AddrOp> is the flexible address operand. See Flexible Address Operand for more information.
* CPSR[Z] is the ALU’s zero status flag as read from the current program status register CPSR.
* CPSR[N] is the ALU’s negative status flag as read from the current program status register CPSR.

### Compare Add (CPA)

The CPA instruction compares two integers by adding them then updating the CPSR

* + - 1. Non-Immediate Syntax

CPA Rd, Rm

* + - 1. Non-Immediate Operation

Rd + Rm, Update CPSR

* + - 1. Immediate Syntax

CPA Rd, #Imm12

* + - 1. Immediate Operation

Rd + Imm12, Update CPSR

Where:

* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Compare Subtract (CPS)

The CPS instruction compares two integers by subtracting them then updating the CPSR.

* + - 1. Non-Immediate Syntax

CPS Rd, Rm

* + - 1. Non-Immediate Operation

Rd – Rm, Update CPSR

* + - 1. Immediate Syntax

CPS Rd, #Imm12

* + - 1. Immediate Operation

Rd – Imm12, Update CPSR

Where:

* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.

### Load Register (LDR)

The LDR instruction loads a word from an offset in read-write data memory and stores it in a register.

* + - 1. Non-Immediate Syntax

LDR Rd, [Rm]

* + - 1. Non-Immediate Operation

Rd <= MEM[Rm]

* + - 1. Immediate Syntax

LDR Rd, [<AddrOp>]

* + - 1. Immediate Operation

Rd <= MEM[<AddrOp>]

Where:

* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.
* MEM is read-write data memory.

### Store Register (STR)

The STR instruction takes a word stored in a register and stores it at an offset in read-write data memory.

* + - 1. Non-Immediate Syntax

STR{N,S} Rd, [Rm]

* + - 1. Non-Immediate Operation

MEM[Rm] <= Rd

* + - 1. Immediate Syntax

STR{N,S} Rd, [<AddrOp>]

* + - 1. Immediate Operation

MEM[<AddrOp>] <= Rd

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* Rm is the first operand register.
* Imm12 is a 12-bit immediate.
* MEM is read-write data memory.

### Push (PSH)

The PSH instruction increments the offset in the stack pointer register SP. Then it takes a word stored in a register and stores it in stack memory at the new offset.

* + - 1. Non-Immediate Syntax

PSH{N,S} Rd

* + - 1. Non-Immediate Operation

SP <= SP + 1

STACK[SP] <= Rd

* + - 1. Immediate Syntax

PSH{N,S} #Imm12

* + - 1. Immediate Operation

SP <= SP + 1

STACK[SP] <= Imm12

Where:

* N is the invert flag. See Optional Suffixes for more information.
* S is the CPSR update flag. See Optional Suffixes for more information.
* Rd is the destination register.
* SP is the stack pointer register.
* STACK is stack memory.

### Pop (POP)

The POP instruction takes the word stored in stack memory at the offset specified by the stack pointer register SP and stores it in a register. Then it decrements the offset in the stack pointer register.

* + - 1. Non-Immediate Syntax

POP Rd

* + - 1. Non-Immediate Operation

Rd <= STACK[SP--]

* + - 1. Immediate Syntax

POP [<AddrOp>]

* + - 1. Immediate Operation

DMEM[<AddrOp>] <= STACK

Where:

* Rd is the destination register.
* SP is the stack pointer register.
* STACK is stack memory.

## ­Table of Instructions (Quick Reference)



Figure .: Complete table of BARNDLES ASM instructions and special terminology.

# Memory

## Memory Encoding

## Memory Image

Because Blender’s shader editor was designed to create shaders and not microprocessors, it is exclusively composed of forward processing logic. In conventional use cases this approach is efficient and effective, but it poses an issue when you’re trying to develop digital memory.

Digital memory requires a feedback loop, a circuitry device that takes signals from one point of a circuit and routes them back into the circuit at an earlier position. Unfortunately, the forward processing nature of Blender’s shader editor makes the implementation of feedback loops impossible, so a work-around had to be developed. The solution is to create a feedback loop outside of the shader editor in an element called the memory image.



Figure .: The feedback loop constructed for the BARNDLES memory system.

In short, the memory image is a 64px by 64px image rendered with Cycles after every clock cycle. Each pixel stores a 24-bit word in memory which means that there is a total of 4,096 words or 12,288 bytes of memory available to the system. Each of these words can be accessed via an address between 0x000 and 0xFFF.

The memory image contains every piece and level of memory available to BARNDLES, so it is divided into sections that serve specific purposes. In its current configuration, the memory image allocates 2,560 words to text memory, 512 words to read-only data memory, 512 words to read/write data memory, 496 words to stack memory, and 16 words to the register file.



Figure .: The sizes and address ranges of the memory image’s sections.



Figure .: Visual display of the sections of the memory image.

In future versions of BARNDLES, it is possible to expand the size of the memory image. Given the system’s 24-bit word size, it can address a up to 224 memory locations with a single word. This rounds out to about 16.8 million words or 50.3 megabytes of potential memory. To take full advantage of this, the memory image would have to be scaled up to a resolution of 4,096px by 4,096px.

The only real challenge in increasing the size of the memory image is the increase in time required to render the memory image using Cycles. The path-tracing nature of Cycles puts a massive bottleneck on the system, so some sort of optimization or workaround will have to be set in place should higher resolution memory images be implemented. Until then, low-resolution memory images will suffice.

## Register File

The BARNDLES register file is composed of 16 registers that each hold a 24-bit word. Like all other memory, the BARNDLES register file is part of the memory image spanning from 0x000 to 0x00F. The registers are organized sequentially within this section of memory such that R0 is stored at 0x000, R1 is stored at 0x001, and so on until we reach the CPSR which is stored at 0x00F.

Of the 16 registers, 12 are general-purpose registers, and the other 4 are system registers.



Figure .: The BARNDLES registers and their functions.

### General-Purpose Registers

BARNDLES has 12 general-purpose registers R0-R11. General-purpose registers are designed to be used and modified by the programmer. This means that no system event will ever modify the values within these registers, apart from system resets.

Although Figure 1.1 details specific functions for general-purpose registers R0-R4, the registers are functionally identical to all other general-purpose registers in the register file. These functions are calling conventions that the programmer can choose to follow if they wish to. Breaking these conventions will have no negative side effects on the architecture nor the execution of instructions.

* + - 1. Return Register

The return register R0 holds return values from subroutines, so they are easily accessible to the calling routine.

* + - 1. Parameter Registers

The parameter registers R1-R4 hold parameter values, so they can be reliably passed into subroutines.

### System Registers

The other 4 registers SP, LR, PC, and CPSR are system registers. Although system registers maintain the same read and write accessibility as general-purpose registers, they should not be written to by the programmer. Many components within the Blender Microprocessor’s architecture are dependent on the values stored in the system registers. Writing to them or otherwise altering their values can have serious side effects on the way the processor executes instructions, determines conditionals, and more.

Each of the system registers have a specific function. Unlike general-purpose registers, these functions are not conventions. They are defined by hardware and must be acknowledged by the programmer.

* + - 1. Stack Pointer

The stack pointer SP stores the offset from the stack base to the position of the last element stored in the stack.

* + - 1. Link Register

The link register LR stores the last value of the program counter when the last branch and link instruction was executed.

* + - 1. Program Counter

The program counter PC stores the offset from the instruction memory base to the position that the next instruction should be fetched from.

* + - 1. Current Program Status Register

The current program status register CPSR stores the ALU status flags C, V, N, and Z.

## Other Memory

The read-write memory section is partitioned into two smaller subsections of memory, stack memory and read-write data memory. These subsections serve specific purposes to the system and the programmer, so their locations, functions, and conventions are defined in full below.

### Stack Memory

Stack memory is a section of memory managed used to temporarily store register values.

Stack memory follows the guidelines of the stack data structure. Elements are pushed and popped from the top of the stack in first-in-last-out order, and elements within the body of the stack are inaccessible unless the elements on top of it are popped out first.

The current offset into stack memory is stored in the Stack Pointer register SP within the BARNDLES register file. Every time a PUSH

While this value is accessible to the programmer, it should not be used to access stack memory manually, nor should it ever be modified by the programmer.

As the name suggests, stack memory is a stack data structure, so elements can only be pushed and popped off the stack in first-in-last-out order. Elements within the stack cannot be randomly accessed.

### Read-Write Data Memory

The system’s read-write data memory (RWDM) is intended to store modifiable data for programs and occupies the memory image from address 0x200 to 0x3FF. RWDM is allocated by the assembler when the program is assembled and cannot be reallocated at runtime, but the values stored within these allocated portions can be written to or otherwise modified.

RWDM should be used to store

### Read-Only Data Memory

The system’s read-only data memory (RODM) is intended to store unmodifiable data for programs and occupies the memory image from address 0x400 to 0x5FF. RODM is allocated by the assembler when the program is assembled and cannot be reallocated nor written to at runtime.

RODM should be used to store unchanging values

### Instruction Memory