Piculet asm sheet	{BHD} can be omitted for word	Word size: 32	Operations with operands that are not 64-bit read the values from LSB.		
Instruction	opcode{type}{regs}{imm_val}	Registers	Description & Effect	Thread Permission	Recorded as command?
Load	LD{B,H,D}{regs}	R1, R2	Load value from address R2 to R1. Omitted type means load a word.		
Store	STR{B,H,D}{regs}	R1, R2	Store value in R1 to address R2. Omitted type means store a word.		
Move	MOV{regs}{imm_val/label}	R1	Moves the immediate value or label into R1.		
Stack push	PUSH{B,H,D}{regs}	R1	Pushes value of R1 to stack.		
Stack pop	POP{B,H,D}{regs}	R1	Pops value from stack into R1.		
Byte swap	BSWAP{H,D}{regs}	R1	Swaps the order of bytes at LSB end of R1.		
Negate	NEG{regs}	R1	Negate all bits in R1 and then increment it by 1.		
Float negate	FNEG{regs}	R1	Toggle the MSB of the 32 bits at LSB end of R1.		
Double negate	DNEG{regs}	R1	Toggle the MSB of R1.		
Add	ADD{D}{regs}	R1, R2, R3	Integer addition R1+R2 assigned to R3.		
Subtract	SUB{D}{regs}	R1, R2, R3	Integer subtraction R1-R2 assigned to R3.		
Multiply	MUL{D}{regs}	R1, R2, R3	Integer multiplication R1*R2 assigned to R3.		
Divide	DIV{D}{regs}	R1, R2, R3	Signed integer division R1/R2 assigned to R3.		
Unsigned divide	UDIV{D}{regs}	R1, R2, R3	Unsigned integer division R1/R2 assigned to R3.		
Modulo	MOD{D}{regs}	R1, R2, R3	Signed integer division R1/R2 remainder assigned to R3.		
Unsigned modulo	UMOD{D}{regs}	R1, R2, R3	Unsigned integer division R1/R2 remainder assigned to R3.		
Add	FADD{regs}	R1, R2, R3	Float32 addition R1+R2 assigned to R3.		
Subtract	FSUB{regs}	R1, R2, R3	Float32 subtraction R1-R2 assigned to R3.		
Multiply	FMUL{regs}	R1, R2, R3	Float32 multiplication R1*R2 assigned to R3.		
Divide	FDIV{regs}	R1, R2, R3	Float32 division R1/R2 assigned to R3.		
Float modulo	FMOD{regs}	R1, R2, R3	Float32 division R1/R2 remainder assigned to R3.		
Power	FPOW{regs}	R1, R2, R3	Float32 exponentiation R1^R2 assigned to R3.		
Add	DADD{regs}	R1, R2, R3	Float64 addition R1+R2 assigned to R3.		
Subtract	DSUB{regs}	R1, R2, R3	Float64 subtraction R1-R2 assigned to R3.		
Multiply	DMUL{regs}	R1, R2, R3	Float64 multiplication R1*R2 assigned to R3.		
Divide	DDIV{regs}	R1, R2, R3	Float64 division R1/R2 assigned to R3.		
Double modulo	DMOD{regs}	R1, R2, R3	Float64 division R1/R2 remainder assigned to R3.		
Power	DPOW{regs}	R1, R2, R3	Float64 exponentiation R1^R2 assigned to R3.		
Compare	CMP{D}{regs}	R1, R2	Integer comparison between R1 and R2.		
Float compare	FCMP{regs}	R1, R2	Float32 comparison between R1 and R2.		
Double compare	DCMP{regs}	R1, R2	Float64 comparison between R1 and R2.		
Float to double	FTOD{regs}	R1 R2	Type conversion from float32 specified by R1 to float64 assigned to R2.		
Double to float	DTOF{regs}	R1 R2	Type conversion from float64 specified by R1 to float32 assigned to R2.		
Integer to float	ITOF{regs}	R1 R2	Type conversion from signed 32-bit int specified by R1 to float32 assigned to R2.		
Integer to double	ITOD{regs}	R1 R2	Type conversion from signed 64-bit int specified by R1 to float64 assigned to R2.		
Float to integer	FTOI{regs}	R1 R2	Type conversion from float32 specified by R1 to 32-bit signed int assigned to R2.		
Double to integer	DTOI{regs}	R1 R2	Type conversion from float64 specified by R1 to 64-bit signed int assigned to R2.		
Left rotate	LROT{regs}	R1, R2	Left rotation of R1 by number of bits specified by R2.		
Right rotate	RROT{regs}	R1, R2	Right rotation of R1 by number of bits specified by R2.		

Left shift	LSH{regs}	R1, R2	Left shift of R1 by number of bits specified by R2.		
Right shift	RSH{regs}	R1, R2	Logical right shift of R1 by number of bits specifed by R2.		
Arith. right shift	ARSH{regs}	R1, R2	Arithmetic right shift of R1 by number of bits specified by R2.		
Bitwise OR	OR{regs}	R1, R2, R3	Bitwise OR of R1 and R2 assigned to R3.		
Bitwise AND	AND{regs}	R1, R2, R3	Bitwise AND of R1 and R2 assigned to R3.		
Bitwise XOR	XOR{regs}	R1, R2, R3	Bitwise XOR of R1 and R2 assigned to R3.		
Register swap	REGWAP{regs}	R1, R2	Swap contents of R1 and R2.		
Register copy	REGCOPY{regs}	R1, R2	Copy contents of R1 to R2.		
Clear	CLR{regs}	R1	Clear contents of register with 0s.		
Fill	FILL{regs}	R1	Fill contents of register with 1s.		
New thread	NTHR{regs}	R1, R2, R3	New thread. R1 is the PC value for the new thread, R2 is address to its info, & its ID is output to R3.	Thread creation	
Destroy thread	DTHR{regs}	R1	Destroys a thread whose ID is specified by the register.		
Detach thread	DTCH{regs}	R1	Detaches a thread whose ID is specified by the register.		
Join thread	JOIN{regs}	R1	Joins a thread whose ID is specified by the register.		
Sleep thread	SLEEP{regs}	R1	Sleeps this thread for a # of nanoseconds that is specified by the register.		
Thread control	THRCTL{regs}	R1, R2, R3	Update/get thread traits. What to update/get specified by R1; new values by R2, and output to R3.	For those updated	
Condition	COND{regs}	R1, R2	Assigns R2 to 1 if the cond. JMP numbered by R1 (0=JMPEQ ... 13=JMPLE) would jump, and 0 if not.		
Jump	JMP{reg/label}	R1	Non-conditional jump to instruction address specified by R1.		
0 Jump if equal	JMPEQ{reg/label}	R1	Jump to instruction address specified by R1 or a label if Z is set.		
1 Jump if not equal	JMPNE{reg/label}	R1	Jump to instruction address specified by R1 or a label if Z is not set.		
2 Jump if C set (left ≥ right)	JMPCS{reg/label}	R1	Jump to instruction address specified by R1 or a label if C is set.		
3 Jump if C clear (left < right)	JMPCC{reg/label}	R1	Jump to instruction address specified by R1 or a label if C is not set.		
4 Jump if negative	JMPN{reg/label}	R1	Jump to instruction address specified by R1 or a label if N is set.		
5 Jump if positive	JMPP{reg/label}	R1	Jump to instruction address specified by R1 or a label if N is not set.		
6 Jump if overflow set	JMPVS{reg/label}	R1	Jump to instruction address specifed by R1 or a label if V is set.		
7 Jump if overflow clear	JMPVC{reg/label}	R1	Jump to instruction address specified by R1 or a label if V is not set.		
8 Jump if uint left > right	JMPHI{reg/label}	R1	Jump to instruction address specified by R1 or a label if C is set and Z is not set.		
9 Jump if uint left ≤ right	JMPLS{reg/label}	R1	Jump to instruction address specified by R1 or a label if C is not set or Z is set.		
10 Jump if signed left ≥ right	JMPGE{reg/label}	R1	Jump to instruction address specified by R1 or a label if N is equivalent to V.		
11 Jump if signed left < right	JMPLT{reg/label}	R1	Jump to instruction address specified by R1 or a label if N is not equivalent to V.		
12 Jump if signed left > right	JMPGT{reg/label}	R1	Jump to instruction address specified by R1 or a label if Z is not set and N is equivalent to V.		
13 Jump if signed left ≤ right	JMPLE{reg/label}	R1	Jump to instruction address specified by R1 or a label if Z is set or if N is not equivalent to V.		
Return	RET	N/A	Sets PC equal to the LR.		
File control	FCTL{regs}	R1, R2, R3	Open/move/create files. Path string address specified by R1, destination string by R2. ID output to R3.	File I/O	
File delete	FDEL{regs}	R1	Delete a file whose file path is given by the string whose address is specified by R1.	File I/O	
File close	FCLOSE{regs}	R1	Close a file whose stream ID is specified by R1 (if 0, closes current file stream).	File I/O	
File set	FSET{regs}	R1	Set current file stream ID to that specified by 16 bits at LSB end of R1.		
File write	FWRITE{regs}	R1, R2, R3	Write to current file; data addressed by R1, file byte offset specified by R2, # bytes specified by R3.	File I/O	
File read	FREAD{regs}	R1, R2, R3	Read current file; output addressed by R1, file byte offset specified by R2, # bytes specified by R3.	File I/O	
File size	FSIZE{regs}	R1, R2	If R1=0, output # bytes of current file to R2. Otherwise, set byte count of current file as specified by R1.	File I/O	
Number of drives	NDRVS{regs}	R1	Output # of avaiable drives to R1.		
List drives	LDRVS{regs}	R1, R2	Writes list of drives to address specified by R1, up to a max # of drives specified by R2.		
File list	FLIST{regs}	R1, R2	Writes list of files & dirs. in file path given by string at address R1, to address specified by R2.		

File list size	FLSIZE{regs}	R1, R2	Outputs to R2 # of bytes FLIST will output given file path string at address R1.		
Set drive	DRVSET{regs}	R1	Sets current drive to ID specified by 8 bits at LSB end of R1.		
Generate object	GEN{regs}	R1, R2, R3	Generates object with type specified by R1, initialization data by R2, and outputs the object's ID to R3.		
Delete object	DEL{regs}	R1	Deletes object with ID specified by R1.		
Bind object	BIND{regs}	R1, R2	Binds object with ID specified by R1; if R1=0, 6 bits at LSB end of R2 must specify object type (bind ID 0)		
Bind framebuffer	BFBO{regs}	R1	Binds FBO with ID specified by R1 to the currently bound command buffer.		
Bind to descriptor	BDSC{regs}	R1, R2	Binds UBO, SBO, TBO, or TLAS specified by R1 to a bound descriptor. R2 is TBO image level if not = 0.		
Bind pipeline	BPIPE{regs}	R1	Binds pipeline object specified by R1		
Update descriptor set	UDSC{regs}	R1, R2	For bound descriptor set, updates # of descriptors specified by R1 with memory addressed by R2.		
Bind set, VBO, or IBO	BSVI{regs}	R1, R2	Bind descriptor set, VBO, or IBO specified by R1. If desc set, set # is specified by 8 bits at LSB end of R2.		Yes
Get buffer size	SIZE{regs}	R1, R2	Outputs to R2 # of bytes in bound object with object type specified by R1.		
Map buffer	MAP{regs}	R1, R2	Maps/unmaps buffer of object specified by R1; if mapping, R2 is set to address of mapped buffer.		
Buffer allocation	BALLOC{regs}	R1, R2	Allocates uninitialized buffer data for an object specified by R1 and # of bytes by R2.		
Update texture	UTEX{regs}	R1, R2	Upload texture to bound TBO; R1 is address to info about update, R2 address to data (in TBO's format).		
Generate mipmaps	GMIPS	N/A	Generates mipmaps for bound TBO.		
Attach to framebuffer	ATTACH{regs}	R1, R2	Set/unset bound TBO level to attachment(s) of bound FBO; R1 specifies attachment, R2 texture level.		
Clear framebuffer	CBUFF{regs}	R1, R2	Clear attachments of the FBO bound at command; R1 specifies attachment, R2 address to clear values		Yes
Update acceleration structure	UACCEL{regs}	R1, R2	Build/update bound AS. R1 specifies action, R2 specifies address to data for update.		Yes
Reset command buffer	RCMD	N/A	Resets the bound command buffer.		
Graphics queue submit	GSUBMIT{regs}	R1	Submit list of command buffers addressed by R1 to a graphics queue.		
Compute queue submit	CSUBMIT{regs}	R1	Submit list of command buffers addressed by R1 to a compute queue.		
Finish commands	FCMDS	N/A	Ends cycle & halts thread until command buffers previously submitted on it have finished.		
Draw call	DRAW{regs}	R1	Draw call with information addressed by R1.		Yes
Indirect draw call	IDRAW{regs}	R1	Indirect draw call with information addressed by R1.		Yes
Buffer update	BUPDATE{regs}	R1	Update data buffer command; information addressed by R1.		Yes
Push constants	PUSHC{regs}	R1	Push constant update command; information addressed by R1.		Yes
Dispatch ray tracing	TRACE{regs}	R1	Dispatches ray tracing operation; information addressed by R1.		Yes
Acceleration structure copy	ASCPY{regs}	R1, R2	Copies acceleration structure whose ID is specified by R1 to that specified by R2.		
Swap buffers	SWAP	N/A	Swap color buffers.		
Display setter	DSET{regs}	R1	Sets the current display ID to that specified by 8 bits at LSB end of R1.		
Set texturing mode	STXMOD{regs}	R1, R2	Sets bound sampler's filtering/wrapping modes. 2 bits at LSB end of R1 is what to update, R2=new value.		
Dispatch compute	DSPCMP{regs}	R1	Dispatches compute operation; information addressed by R1.		Yes
Get object bindings	GETBND{regs}	R1	Outputs to R1 address of the beginning of object bindings.		
Get hardware info	GETHWI{regs}	R1	Outputs to R1 address of the beginning of hardware information.		
Clear 7 bytes	CLR7{regs}	R1	Clear upper 7 bytes of R1 to 0s.		
Clear 6 bytes	CLR6{regs}	R1	Clear upper 6 bytes of R1 to 0s.		
Clear 4 bytes	CLR4{regs}	R1	Clear upper 4 bytes of R1 to 0s.		
Sign extension; upper 7 bytes	SEXT7{regs}	R1	Fill upper 7 bytes of R1 with 1s if MSB of the byte at the LSB end of R1 is 1, and with 0s otherwise.		
Sign extension; upper 6 bytes	SEXT6{regs}	R1	Fill upper 6 bytes of R1 with 1s if MSB of the 2 bytes at LSB end of R1 is 1, and with 0s otherwise.		
Sign extension; upper 4 bytes	SEXT4{regs}	R1	Fill upper 4 bytes of R1 with 1s if MSB of the 4 bytes at LSB end of R1 is 1, and with 0s otherwise.		
Update segments	USEG{regs}	R1, R2, R3	Updates segments in bound segment table object. R1 specifies what to do, and R2 the segment entry ID.		
Bitwise negate	BITN{regs}	R1	Negate all bits in R1.		
Increment	INCR{regs}	R1	Increment R1 by 1.		

Decrement	DECR{regs}	R1	Decrement R1 by 1.		
Math function	MATHF{regs}	R1, R2, R3	Calculates result of single-variable math function named by R1 on R2 and outputs to R3.		
Display/image capture	DCAPT{regs}	R1, R2	Writes a copy of the current display/camera image (specified by R1) to address R2.	Capture permission	
Time	TIME{regs}	R1, R2	Outputs some aspect of current time info specified by R1 to R2.		
Memory copy	MCOPY{regs}	R1, R2, R3	Writes to address R3 a copy of memory addressed by R1 for a number of bytes specified by R2.		
Audio source/listener control	ASLCTL{regs}	R1, R2, R3	Gets/sets info for audio src/listeners. R1=what to do, R2=data to set, R3=where to output data.		
Audio data/file control	ADFCTL{regs}	R1, R2, R3	Gets/sets info for audio data/files. R1=what to do, R2=data to set, R3=where to output data.		
Video data/file control	VDFCTL{regs}	R1, R2, R3	Gets/sets info for video data/files. R1=what to do, R2=data to set, R3=where to output data.		
Network control	NETCTL{regs}	R1, R2	Performs networking operations/gets networking info. R1=what to do, R2=data for operation.	Networking	
Store data left SIMD vector	SLVEC{regs}	R1, R2	Stores data to left SIMD vector. R1 specifies number of bytes, R2 addresses the data to store.		
Store data right SIMD vector	SRVEC{regs}	R1, R2	Stores data to right SIMD vector. R1 specifies number of bytes, R2 addresses the data to store.		
SIMD operation	SIMD{regs}	R1, R2	Performs SIMD operation. 8 bits at LSB end of R1 specifies what to do; writes results to address R2.		