TLB miss and audio / video co-processor

Qupls4 System Control Processor

Robert Finch

Table of Contents

Qupls4_co-processor

Overview

The Qupls4 co-processor is co-processor that handles MMU, display and audio controller tasks. It can update the display controller's register set at specific points during the display generation. The co-processor has a small instruction set. Co-processor instructions are 32-bit words. All data values are 64-bit.

If enabled, during every vertical reset of the display the co-processor's instruction pointer is set to point at location 10h and the c-processor begin executing instructions.

For TLB misses the co-processor begins executing instructions at location 008h.

Interrupts

The co-processor supports three sources of interrupts, the video frame interrupt, TLB miss interrupt and graphics queue not empty interrupt. Co-processor interrupts are very high speed. It takes three clock cycles to re-awaken from low power mode. Registers are swapped in one of those three cycles.

A TLB miss interrupt will automatically disable paging. Paging will be enabled on return from the interrupt subroutine.

Operation

Normal operation is to execute a WAIT instruction which will put the co-processor in a lower power mode, then process the interrupt when it occurs.

When the interrupt occurs the instruction pointer for the instruction is stored on an internal stack. Then registers r1 to r8 are also placed on the internal stack. On interrupt return the registers are restored and the instruction pointer is loaded from the internal stack.

Vectors

The co-processor vectors to the following locations on reset or interrupt.

Vector	Value
Reset	$0000
TLB Miss	$0008
Vertical Sync	$0100

Priority

TLB miss interrupts have a higher priority than display interrupts. TLB interrupts are allowed to interrupt the display interrupt processing routine.

Co-processor Programming Model

There are fifteen 64-bit general purpose registers r1 to r15. R0 is always zero.

Internal Stack

There is a 16-entry stack which can hold the IP plus registers r1 to r8. The stack is not directly accessible.

When an interrupt occurs or a subroutine call is done (CALL) the instruction pointer and the first eight registers are all placed on the internal stack in a single clock cycle. A two-bit flag is also saved on the stack indicating the call type (subroutine TLB miss, or frame interrupt).

On a return (RET) instruction the stack is popped and the IP and first eight GPRs may be restored in a single clock cycle. Which registers get restored depends on a register list mask specified by the RET instruction. For interrupts the mask should be set to FFh to restore all eight registers.

Co-processor Control Registers

Co-processor control registers are located at the upper end of the address space.

	63 … 0			
$7F60		F	E	Control Register
$7F80	Page Table Base Address – TLB #0			
$7F90	Page Table Attributes – TLB #0			
$7FA0	Page Table Base Address – TLB #1			
$7FB0	Page Table Attributes – TLB #1			

Co-processor Control Register

This register has bits to enable the co-processor. It also has a bit indicating the restart rate for the co-processor. The co-processor program may be automatically restarted at the beginning of every video frame (the default configuration) or it may restart every 16th frame.

Co-processor Local Memory Space

The co-processor has a small 64kB BRAM memory that is shared between the co-processor and the system.

Command Register

The command code register specifies which graphics operation to perform. Write to the queue trigger register ($DC8) to queue the command.

Summary of Graphics Commands

$Command_8$	Operation Performed	Parameters / Set first
0	Draw character bitmap	X0,Y0 fgcolor, bkcolor, char code
1	Plot point	X0,Y0 bkcolor
2	Draw line	x0,y0 x1,y1 bkcolor
3	Draw filled Rectangle	x0,y0 x1,y1 bkcolor
6	Draw filled triangle	x0,y0 x1,y1 x2,y2, bkcolor
8	Draw Bezier curve	x0,y0 x1,y1 x2,y2, bkcolor, fill code
12	Set pen color	RGB888 value
13	Set fill color	RGB888 value
14	Set alpha	16-bit value
16	Set X0	32-bit fixed point (16,16) value
17	Set Y0	32-bit fixed point (16,16) value
18	Set Z0	32-bit fixed point (16,16) value
19	Set X1	32-bit fixed point (16,16) value
20	Set Y1	32-bit fixed point (16,16) value
21	Set Z1	32-bit fixed point (16,16) value
22	Set X2	32-bit fixed point (16,16) value
23	Set Y2	32-bit fixed point (16,16) value
24	Set Z2	32-bit fixed point (16,16) value
25	Set Clip X0	16-bit value (whole number)
26	Set Clip Y0	16-bit value (whole number)
27	Set Clip X1	16-bit value (whole number)
28	Set Clip Y1	16-bit value (whole number)
29	Set clip enable / disable	1 bit value
32	Set aa transform coefficient	32-bit fixed point (16,16) value
33	Set ab transform coefficient	32-bit fixed point (16,16) value
34	Set ac transform coefficient	32-bit fixed point (16,16) value
35	Set at transform coefficient	32-bit fixed point (16,16) value
36	Set ba transform coefficient	32-bit fixed point (16,16) value
37	Set bb transform coefficient	32-bit fixed point (16,16) value
38	Set bc transform coefficient	32-bit fixed point (16,16) value
39	Set bt transform coefficient	32-bit fixed point (16,16) value
40	Set ca transform coefficient	32-bit fixed point (16,16) value
41	Set cb transform coefficient	32-bit fixed point (16,16) value
42	Set cc transform coefficient	32-bit fixed point (16,16) value
43	Set ct transform coefficient	32-bit fixed point (16,16) value
44		32-bits
45		
46		
254	Reset command queue	
255	NOP	

Command #0 – Draw Character Bitmap

Command #0 will draw the character specified by the low order 16-bits of the parameter portion of the command holding register at previously set X0, Y0 co-ordinate registers, using a previously set foreground and background color.

Command #1 – Plot Point

Command #1 will plot a point on the target bitmap using previously set X0, Y0 co-ordinates in the previously set background color. The raster operation used to set the point is determined from bits 4 to 7 of the command parameter.

Command #2 – Draw Line

Command #2 will draw a line on the target bitmap using the previously set X0, Y0, X1, and Y1 co-ordinates in the previously set background color. The raster operation used is determined from bits 4 to 7 of the command parameter.

Command #3 – Draw Filled Rectangle

Command #3 will draw a filled rectangle on the target bitmap using the previously set X0, Y0, X1, and Y1 co-ordinates in the previously set background color. The raster operation used is determined from bits 4 to 7 of the command parameter.

Command #6 – Draw Triangle

Command#6 will draw a filled triangle on the target bitmap using the previously set X0, Y0, X1, Y1, and X2, Y2 co-ordinates. The raster operation used is determined from bits 4 to 7 of the command parameter.

Command#8 – Draw Bezier Curve

Command#8 will draw a filled or unfilled Bezier curve on the target bitmap using previously set co-ordinates. The least significant two bits of the command parameter determine how the curve is filled. The raster operation used is determined from bits 4 to 7 of the command parameter.

Command #12 – Set Pen Color

This command will set the graphics pen color for subsequent operations using the pen color. The pen color set must be a RGB888 value.

Command #13 – Set Fill Color

This command will set the fill color for subsequent operations using the fill color. The fill color set must be an RGB888 value.

Command #14 – Set Alpha

This command sets the alpha value for subsequent operations using an alpha value. The alpha value is a 16-bit number.

Command #16 – Set X0

This command sets the graphics X0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #17 – Set Y0

This command sets the graphics Y0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #18 – Set Z0

This command sets the graphics Z0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #19 – Set X1

This command sets the graphics X1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #20 – Set Y1

This command sets the graphics Y1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #21 – Set Z1

This command sets the graphics Z1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #22 – Set X2

This command sets the graphics X2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #23 – Set Y2

This command sets the graphics Y2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #24 – Set Z2

This command sets the graphics Z2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

Command #25 – Set ClipX0

This command sets the clipping region X0 co-ordinate. The parameter value is a 16-bit integer (whole number only).

Command #26 – Set ClipY0

This command sets the clipping region Y0 co-ordinate. The parameter value is a 16-bit integer.

Command #27 – Set ClipX1

This command sets the clipping region X1 co-ordinate. The parameter value is a 16-bit integer (whole number only).

Command #28 – Set ClipY1

This command sets the clipping region Y1 co-ordinate. The parameter value is a 16-bit integer.

Command #29 – Set Clip Enable / Disable

This command determines whether clipping of graphics output is enabled or disabled. If the parameter is 1 clipping is enabled, if the parameter is 0 then clipping is disabled. Note that all graphics are automatically clipped according to the target width and height which cannot be disabled. This setting controls the clip region defined by clipping co-ordinates clipx0, clipy0, clip x1, clip y1.

Command #254 – Reset Command Queue

This command empties out the command queue. Any command in the queue are not performed.

Command #255 – NOP

This command is a no-operation.

Fonts

The controller features a dedicated text blitter that draws text on the screen. It can handle either fixed or varying width fonts. Multiple fonts may be supported through the use of font tables.

Font Table

The core supports the use of multiple fonts onscreen at the same time via a font table. The font table is a table of information describing basic characteristics of the font and where to find further font information for a given number of fonts. The font table is in memory and indexed by the font id register to select a font to work with. The font table is a collection of font table entries each of which has the following layout:

Font Table Entry

Offset	Fields				Use
0	$Address_{31..0}$				Address of character bitmaps
4	$fixed_1$	$width_5$	$height_5$	$\sim_{21}$	width and height
8	$Address_{31..0}$				Glyph width table address
C	$\sim_{32}$				

Each font table entry is sixteen bytes in size. The font id (register $DE8) is used to index into this table. Setting the font table id register tells the core which font to use. The core then looks up the font information from the table.

The location of the font table in the controller's memory is specified in the font table address register - register $DE0.

Glyph Width Table

If the font is a fixed width font then no further table lookup is required, and the font width is determined by the width field in the font table entry. For fonts with characters whose bitmaps vary in width there is an additional table used to describe the width of the bitmap for the character in memory. Thus, variable width character fonts are supported.

Font Table Address

This register determines where in the controller's memory the font table is located.

The glyph width table is an array of bytes that specify the character width for each character in the font. The address of the glyph width table is found in the font table entry for the font.

	63		0	
$DE0	$\sim_{32}$	$Offset_{31..0}$		font table address
$DE8	$\sim_{48}$		$Font\ id_{16}$	

Font ID

This register selects the working font. The core uses this id to determine which entry of the font table to use to lookup font attributes.

Blitter

Overview

The SCP has a powerful blitter component which may be used to transfer information in the controller's memory extremely fast. The blitter consists of four DMA channels (A, B, C, and D). A, B, and C are data source channels and D is a data destination channel. The destination channel may be used in a standalone fashion to draw lines or fill areas. Any or all three of the source channels may be active to fetch data to transfer to the destination. A variety of operations between the data fetched by channels A, B, and C are possible including copy and masking operations.

	31 … 0	
$D00	$Address_{31..0}$	Channel A address low
$D04	$\sim_{32}$	Channel A address high
$D08	$Offset_{31..0}$	Channel A modulo low
$D0C	$\sim_{32}$	Channel A modulo high
$D10	$Count_{32}$	Channel A Count
$D14	$\sim_{32}$	
$D20	$Address_{31..0}$	Channel B address low
$D24	$\sim_{32}$	Channel B address high
$D28	$Offset_{31..0}$	Channel B modulo low
$D2C	$\sim_{32}$	Channel B modulo high
$D30	$Count_{32}$	Channel B Count
$D34	$\sim_{32}$	
$D40	$Address_{31..0}$	Channel C address low
$D44	$\sim_{32}$	Channel C address high
$D48	$Offset_{31..0}$	Channel C modulo low
$D4C	$\sim_{32}$	Channel C modulo high
$D50	$Count_{32}$	Channel C Count
$D54	$\sim_{32}$	
$D60	$Address_{31..0}$	Channel D address low
$D64	$\sim_{32}$	Channel D address high
$D68	$Offset_{31..0}$	Channel D modulo low
$D6C	$\sim_{32}$	Channel D modulo high
$D70	$Count_{32}$	Channel D Count
$D74	$\sim_{32}$	
$D78	$\sim_{16}$ | $Data_{16}$	Channel D Data
$D7C	$\sim_{32}$	
$D80	$BltSrcWid_{32}$	Source width
$D84	$\sim_{32}$	
$D88	$BltDstWid_{32}$	Destination width

$D8C	~32			
$D90	~16		Op16	Blitter operation code
$D94	~32			
$D98	~2	PLD6	~8	BltCtrl16	Blitter Control
$D9C	~32			

Channel A Address

This pair of registers set the address of data source for channel A.

Channel A Modulo

This pair of registers set the modulo amount for channel A. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

Channel A Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

Channel B Address

This pair of register sets the address of data source for channel B.

Channel B Modulo

This pair of registers set the modulo amount for channel B. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

Channel B Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

Channel C Address

This pair of register sets the address of data source for channel C.

Channel C Modulo

This pair of registers set the modulo amount for channel C. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

Channel C Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

Channel D Address

This pair of registers set the address of data destination for channel D.

Channel D Modulo

This pair of registers set the modulo amount for channel D. The modulo amount is an amount added to the current working address once transfers have hit the destination width specification.

Channel D Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

Source Width

The source width specifies the number of horizontal pixels in the bitmap. It is used along with the modulo register to calculate the address of the bitmap data for a source.

Destination Width

The destination width specifies the number of horizontal pixels in the bitmap. It is used along with the modulo register to calculate the address of the bitmap data for a source. As an example the screen bitmap is 800 pixels wide, so the width value placed in the register would be 800.

Blit Control

This register contains bits for control of the blit operation. Channels may be independently enabled or disabled. They may also be set to descending mode where the address decrements through memory instead of incrementing.

Bit	Default	Purpose		
0	0	Channel A bitmap mode enable		
1	0	Channel A DMA enable		
2	0	Channel B bitmap mode enable		
3	0	Channel B DMA enable		
4	0	Channel C bitmap mode enable		
5	0	Channel C DMA enable		
6	0	reserved		
7	0	reserved		

8	0	Channel A descend mode		
9	0	Channel B descend mode		
10	0	Channel C descend mode		
11	0	Channel D descend mode		
12	0	reserved		
13	1	Blit done indicator		
14	0	Blit active indicator		
15	0	Blit operation trigger		

Hardware Cursor Control

The controller has 32 hardware cursors. The cursors may be up to 32 pixels wide and 512 pixels high. The cursors all share a common 256 entry color palette. However, each cursor may have its own set of colors. A cursor by itself may use three different colors simultaneously plus transparency. Note that the cursors are effectively 32 pixels wide, but pixels may be set to be transparent, so the apparent size of the cursor looks smaller.

Cursor Color Palette

The cursor color palette has 256 entries each of which is a 64-bit vector including additional attributes besides just the color. Attributes include alpha blending, reverse video and flashing.

The registers are organized into groups of four, a group of four registers present for each of the cursors. Thus, each cursor can have a different set of colors from other cursors. Only three of the four registers are used. (The color code 00 is transparent). The registers are further organized into sixteen groups of sixteen for linked sprites. A set of 16 color registers is used when sprites are linked together. The first group establishes a set of colors which are shared between sprite 0 and 1. The second group is shared between sprites 2 and 3, and so on. Note that color palette entry #0 is never used.

$000	$\sim_{26}$	i	f	$rate_4$	$Alpha_8$	$RGB888_{24}$	Color0	
$008 to $7F8	255 more registers							

$Alpha_8$ determines how much of the cursor color is present in the output. A value of zero causes the cursor to be fully output. A value of all ones will make the cursor invisible. Alpha blending can be used to create shadows by selecting a cursor color of black then setting the alpha register to a none-zero value.

I - The I flag indicates to reverse the video output. The color under the cursor is xor'd with -1.

f – indicates to flash the cursor. The cursor will flash at a rate determined from the $rate_4$ field.

Cursor Link Register

The cursor link register indicates cursors which are linked to the next cursor to increase the apparent number of colors available to sixteen rather than four. Each bit in the register specifies the link state for the corresponding sprite. Linked cursors must have their coordinates maintained with the same values.

$F68	$\sim_{32}$	$Link_{31..0}$

Cursor Enable Register

The cursor enable register controls which sprites are visible on the screen. Bits in the register enable the sprite display when set to a one for the sprite corresponding to the bit number.

$F60	~32	Enable31..0

Collision Register

The collision register indicates which sprites are colliding with other sprites. Each bit in the register corresponds to a sprite. If the bit is set then the sprite has collided with another sprite. The bits will remain set in the register until the register is updated.

$F70	~32	SprCollision31..0

Cursor Control Registers

The control register layout for all cursors is identical. The layout is shown only for the first cursor, cursor #0. Note that the count and position registers are 16-bit addressable. Any or all of the 16-bit fields may be updated.

$800	Address31..0				bitmap address low
$804	(Reserved) Address63..32				bitmap address high
$808	zpos8	~16	MCnt16		total number of pixels
$80C	~4	vpos12	~4	hpos12	vertical / horizontal position
$810 to $81C	Cursor #1 Registers				
...	...				
$9E0 to $9FC	Cursor #31 Registers				
$A00 to $BFC	reserved				

Cursor Bitmap Address

This register contains the address of the cursor's bitmap in the core's memory. The cursor bitmap occupies contiguous words of memory. The amount of memory required is determined by the MCnt field for the cursor. Each raster line of the cursor is composed of one sixty-four-bit value to allow bitmaps up to 32 pixels in width to be defined. The amount of memory required is one word, which is a fixed amount. Even if the cursor is only five pixels wide, a whole word of memory per scanline is still required.

MCnt

The size register controls the visible size of the cursor. Cursors may be up to 32 pixels in size horizontally, and up to 512 scan lines vertically. The value placed in the register should

be one less than the total count of pixels to display. The count should be a multiple of 32 pixels then minus one. For example, for a 32hx30v sprite the count would be 960-1 = 959.

Horizontal and Vertical Position

The horizontal and vertical position are relative to the top left corner of the visible screen which is position (0, 0).

Z-Order

The z-order (zpos) register controls the appearance priority of the cursor compared to other graphics on-screen. If the cursor's z-order is less than the z-order of the current pixel it will appear in front of the pixel. Otherwise it will be hidden by the pixel.

Co-processor Instruction Set Description

WAITcc

Opcode: 1

Description:

The wait instruction waits for a signal combination or interrupt to occur. This could be the display generation scan to reaching a specific horizontal, vertical and frame position. The co-processor is not active while waiting and other devices may freely access the display register set.

The co-processor will also wait for a write cycle to occur at the specified local memory address. The value written will be loaded into the destination register Rd. The local memory address must be within the last 2k words of the address space. This address can thus be used as a signal by an external processor.

While waiting the co-processor is placed into a lower power mode. Three clock cycles are required to exit the mode.

Instruction Format:

Bits	Field	Description
21:31	Address	Signal address last 2k
17:20	Cond	condition
13:16	Rs2	Signal conditions mask
9:12	Rs1	Signal conditions value
5:8	Rd	Command value
4:0	1	WAIT opcode

Cond – condition and operation to perform

$Cond_4$	cc	Jump operation
0 to 7		Reserved
9	GEP	Scan position after {Rs1, Rs2}
10		Unconditional WAIT
9 to 14		reserved

Signal Conditions:

Rs1:

Bits	Field	Description
63:60	~	reserved
48	B	blitter wait

37:32	F	frame. to wait for
27:16	V	vert. pos. to wait for
11:0	H	horiz. pos. to wait for

Rs2

Bits	Field	Description
11:0	H	horiz. pos. to wait for
37:32	MF	mask: frame
27:16	MV	mask: vertical pos
11:0	MH	mask: horizontal pos

B – indicates that the co-processor should wait for an outstanding blit operation to complete before continuing.

F – the frame number that the co-processor should wait for. This may be masked off by the frame mask (MF) field.

V – the vertical position that the co-processor should wait for. This may be masked by the vertical position mask field (MV)

H – the horizontal position that the co-processor should wait for. this may be masked by the horizontal position mask field (MH).

LOAD

Opcode: 16

Description:

The LOAD instruction moves a value from a register or memory to Rd. The load address is specified as the sum of Rs1 and a sign extended displacement constant.

Instruction Format:

Bits	Field	Description
17:31	Disp	Displacement
13:16	~	reserved
9:12	Rs1	Base Address
5:8	Rd	Destination register (not used)
4:0	16	STORE opcode

STORE

Opcode: 17

Description:

The STORE instruction moves a value from Rs2 into one of the display controller or MMU registers specified as the sum of Rs1 and a sign extended displacement constant. This allows the co-processor to do things like initiate a blitter operation or trigger an interrupt.

Instruction Format:

Instruction Format:

Bits	Field	Description
17:31	Disp	Displacement
13:16	Rs2	Value to store
9:12	Rs1	Base Address
5:8	Rd	Destination register (not used)
4:0	17	STORE opcode

STOREI

Opcode: 18

Description:

The STORE instruction moves an immediate value into one of the display controller or MMU registers specified as the sum of Rs1 and a sign extended displacement constant. This allows the co-processor to do things like initiate a blitter operation or trigger an interrupt.

If the immediate is larger than 14 the immediate constant follows the instruction.

Instruction Format:

Bits	Field	Description
17:31	Disp	Displacement
13:16	Rs2	Value to store
9:12	Rs1	Base Address
5:8	Rd	Immediate 0 to 14
4:0	18	STORE opcode

Jcc

JEQ / JNE / JLT / JLE / JGE / JGT / JLEP / JGEP / DJNE

Opcode: 4

Description:

This instruction conditionally jumps to a destination address if the relationship between Rs1 and Rs2 is true.

The DJNE instruction decrements the Rs1 register then jumps if the decremented value is not equal to Rs2.

Instruction Format:

Bits	Field	Description
17:31	Address	
13:16	Rs2	Test register (mask)
9:12	Rs1	Test register (value)
5:8	Cond	Branch condition
4:0	4	Jcc opcode

Cond – condition and operation to perform

$Cond_4$	cc	Jump operation
0	EQ	Rs1 = Rs2
1	NE	Rs1 <> Rs2
2	LT	Rs1 < Rs2
3	LE	Rs1 <= Rs2
4	GE	Rs1 >= Rs2
5	GT	Rs1 > Rs2
6	DJNE	Rs1 = Rs1 -1, Rs1 != Rs2
8	LEP	Scan position before {Rs1, Rs2}
9	GEP	Scan position after {Rs1, Rs2}
10	GQE	Graphics que empty
11	GQNE	Graphics que not empty
12 to 15		reserved

JUMP

Opcode: 9

Description:

This instruction unconditionally jumps to a destination address. The destination address is the sum of Rs1 and a sign extended constant value.

Instruction Format:

Bits	Field	Description
17:31	address	
13:16	~	Reserved
9:12	Rs1	Base address
5:8	0	Op
4:0	9	JMP opcode

JSR

Opcode: 9

Description:

This instruction unconditionally jumps to a subroutine by pushing the return address and registers r1 to r8 on an internal stack then changing program flow to the destination address. The destination address is the sum of Rs1 and a sign extended constant value.

Registers may be optionally popped from the stack by the RET instruction.

Instruction Format:

Bits	Field	Description
17:31	address	
13:16	~	Reserved
9:12	Rs1	Base address
5:8	1	Op
4:0	9	JMP opcode

RET

Opcode: 9,2

Description:

This instruction unconditionally returns from a subroutine or interrupt by popping the return address. Registers r1 to r8 may be popped off an internal stack according to a register list mask. Bit 0 of the mask represents r1, bit 1 represents r2, and so on.

Instruction Format:

Bits	Field	Description
17:31	Reglist	Which registers to restore
13:16	~	Reserved
9:12	~	Reserved
5:8	2	Op
4:0	9	JMP opcode

CALC_INDEX

Opcode: 12

Description:

This instruction computes the page table index or the PTE for the current miss address using the page size from the page table attributes register and the level supplied by register Rs1 and places the value in register Rd. Note the calculated value still needs to be masked according to the number of entries in the page table. This is done by CALC_ADR.

This instruction replaces about four operations with an operation executed in a single cycle.

Calculation:

Index = miss_address >> ((log2(page size)– 3) * level + log2(page size))

Instruction Format:

Bits	Field	Description
17:31	~	reserved
13:16	~	Reserved
9:12	Rs1	Table level – three LSB bits
5:8	Rd	Index value
4:0	12	opcode

CALC_ADR

Opcode: 13

Description:

This instruction computes the address of the PTE for the current miss address using the page size from the page table attributes register, the index into the page table supplied by CALC_INDEX in register Rs2 and the base address of the table supplied by register Rs1 and places the value in register Rd.

This instruction replaces about four operations with an operation executed in a single cycle.

Calculation:

Mask = (1 << (log2(page size)) - 1

Address = Rs1 | (Rs2 & mask)

Instruction Format:

Bits	Field	Description
17:31	~	reserved
13:16	Rs2	Table index (from CALC_INDEX)
9:12	Rs1	Page table address
5:8	Rd	Address of PTE
4:0	13	opcode

BUILD_ENTRYNO

Opcode: 14

Description:

This instruction builds the entry_no argument required to access the TLB. It takes the PTE index in Rs1, the way in Rs2 and a constant used to determine whether to read or write the PTE.

This instruction replaces about three operations with an operation executed in a single cycle.

Calculation:

Entry_no = Rs1[15:0] | (Rs2[7:0] << 16) | (1 << const)

Instruction Format:

Bits	Field	Description
17:31	Const	Only the lower five bits used
13:16	Rs2	way
9:12	Rs1	PTE index
5:8	Rd	Entry_no
4:0	14	opcode

BUILD_VPN

Opcode: 15

Description:

This instruction builds the VPN portion of a TLB entry required to update the TLB. It takes the current miss address, asid, and current count and builds it into a single value.

This instruction replaces about four operations with an operation executed in a single cycle.

Calculation:

VPN = (miss address) >> (log2(page size) + log2(TLB entries)) | (asid << 48) | (count << 42)

Instruction Format:

Bits	Field	Description
17:31	~	Reserved
13:16	~	Reserved
9:12	~	Reserved
5:8	Rd	VPN, ASID, COUNT
4:0	15	Opcode

FLUSH

Opcode: 8

Description:

This instruction may be used to trigger or enable a TLB flush either by ASID or for the entire TLB. The status of the flush is returned in register Rd.

Instruction Format:

Bits	Field	Description
19:31	~	reserved
18	Trig	1=trigger flush
17	En	1=enable flush
13:16	~	Reserved
9:12	Rs1	ASID to flush, 0=entire TLB
5:8	Rd	Bit 63 = done status, 1 = done
4:0	8	Opcode

ADD

Opcode: 22

Description:

This instruction adds Rs1, Rs2 and a sign extended constant. This instruction may also be used to load a constant into a register.

Calculation:

Rd = Rs1 + Rs2 + Imm

Instruction Format:

Bits	Field	Description
17:31	Imm	Constant
13:16	Rs2	2nd operand
9:12	Rs1	1st operand
5:8	Rd	Result value
4:0	22	Opcode

ADD64

Opcode: 5

Description:

This instruction adds Rs1, Rs2 and a 64-bit constant. The immediate constant follows the instruction. This instruction may also be used to load a 64-bit constant into a register. This instruction takes two or three clock cycles depending on the alignment of the constant.

Calculation:

Rd = Rs1 + Rs2 + Imm

Instruction Format:

Bits	Field	Description
17:31	~	reserved
13:16	Rs2	2nd operand
9:12	Rs1	1st operand
5:8	Rd	Result value
4:0	5	Opcode

AND

Opcode: 24

Description:

This instruction bitwise ‘ands’ Rs1, Rs2 and a sign extended constant.

Calculation:

Rd = Rs1 & Rs2 & Imm

Instruction Format:

Bits	Field	Description
17:31	Imm	constant
13:16	Rs2	2nd operand
9:12	Rs1	1st operand
5:8	Rd	Result value
4:0	24	Opcode

AND64

Opcode: 23

Description:

This instruction bitwise ‘ands’ Rs1, Rs2 and a 64-bit constant. The immediate constant follows the instruction. This instruction takes two or three clock cycles depending on the alignment of the constant.

Calculation:

Rd = Rs1 & Rs2 & Imm

Instruction Format:

Bits	Field	Description
17:31	~	reserved
13:16	Rs2	2nd operand
9:12	Rs1	1st operand
5:8	Rd	Result value
4:0	23	Opcode

OR

Opcode: 25

Description:

This instruction bitwise ‘ors’ Rs1, Rs2 and a sign extended constant.

Calculation:

Rd = Rs1 | Rs2 | Imm

Instruction Format:

Bits	Field	Description
17:31	Imm	constant
13:16	Rs2	2nd operand
9:12	Rs1	1st operand
5:8	Rd	Result value
4:0	25	Opcode

PLOT

Opcode: 10

Description:

This instruction plots a point at the graphics location specified by Rs1 and Rs2.

Instruction Format:

Bits	Field	Description
17:31	~	reserved
13:16	Rs2	Y co-ordinate
9:12	Rs1	X co-ordinate
5:8	~	Reserved
4:0	10	Opcode

XOR

Opcode: 26

Description:

This instruction bitwise exclusive ‘ors’ Rs1, Rs2 and a sign extended constant.

Calculation:

Rd = Rs1 ^ Rs2 ^ Imm

Instruction Format:

Bits	Field	Description
17:31	Imm	constant
13:16	Rs2	2nd operand
9:12	Rs1	1st operand
5:8	Rd	Result value
4:0	26	Opcode

SHL

Opcode: 20

Description:

This instruction performs a left shift operation on Rs1 using a count which is the bitwise or of the value in Rs2 and the immediate. Usually either the count or Rs2 would be zero. The shift count is limited to five bits allowing shifts of up to 31 bits.

Calculation:

Rd = Rs1 << (Rs2 | imm)

Instruction Format:

Bits	Field	Description
17:31	Imm	Five LSBs only
13:16	Rs2	Count
9:12	Rs1	Operand to shift
5:8	Rd	Result value
4:0	20	Opcode

SHR

Opcode: 21

Description:

This instruction performs a right shift operation on Rs1 using a count which is the bitwise or of the value in Rs2 and the immediate. Usually either the count or Rs2 would be zero. The shift count is limited to five bits allowing shifts of up to 31 bits.

Calculation:

Rd = Rs1 >> (Rs2 | imm)

Instruction Format:

Bits	Field	Description
17:31	Imm	Five LSBs only
13:16	Rs2	Count
9:12	Rs1	Operand to shift
5:8	Rd	Result value
4:0	21	Opcode