Gambit

Overview

Gambit is a superscalar processor with a 52-bit native operating mode. Native mode makes use of a 32-entry register file. Native mode instructions vary in length up to 52-bits. The processor manages branching using a compare instruction and compare result registers.

Motivation

Gambit's goal is to be a cost reduced processing core. Consuming 20% fewer resources than a 64-bit core would.

Programming Model

General Registers

Gambit contains a 32-entry by 52-bits wide general-purpose integer register file. One register designation is reserved for the stack pointer. There are four stack pointers, one for each operating level.

Reg			Usage
R0	z	This register is always zero	
R1	acc	Accumulator	First parameter / return value / loop count
R2	x	'x' index register	Second parameter
R3	y	'y' index register	Third parameter
R4			
R5			
R6			
R7 to 29			
R30	fp	frame pointer	
R31	sp	stack pointer	user stack pointer
R31	ssp		supervisor stack pointer
R31	hsp		hypervisor stack pointer
R31	msp		machine stack pointer

Compare Result Registers

Compare result registers are two-bit registers that can hold a value from -1 to 1. Compare results are set as the target to several instructions including compare, bit test, and set operations.

-1 means the first operand was less than the second one. 0 means both operands are equal, +1 means the first operand was greater than the second.

Reg	Usage Convention
C0	Integer Compares / set
C1	Float compares / set
C2 to C7	

Link Registers

There are four link registers provided for subroutine calls. There is an additional link register to support exception processing routines.

Reg	Usage
L0	scrap (jumps)
L1	subroutine calls
L2	
L3	

XL	exception link

Register Tags

As part of the internal workings of the core, it tracks registers using a seven bit register tag associated with each register in the programming model.

Tag	Register	
0 to 31	r0 to r31 general-purpose integer registers	
32 to 63	f0 to f31 general-purpose floating-point registers	
64 to 95	reserved for vector registers	
96 to 100	link registers	
101	machine status register	
103	packed compare results	
104 to 111	compare results registers	
112 to 119	reserved for vector mask registers	

Status Register

51 37	36 29	28	27 26	25 24	23 14	13 4	3	2	1	0
~	ASID	MPRV	XS	FS	DLS	OLS	~	I_3		

Bit		Meaning
0 to 2	I_3	interrupt mask
4 to 13	OLS	operating level stack (bits 4 to 5 = current operating level)
14 to 23	DLS	data level stack (bits 14 to 15 = current data level)
24 to 25	FS	floating point state
26 to 27	XS	additional core extension state
28	MPRV	memory data level swap
29 to 36	ASID	address space identifier
37 to 51		reserved

Control and Status Registers

Control Register Zero (CSR #000)

This register contains miscellaneous control bits including a bit to enable protected mode.

Bit		Description
0	Pe	Protected Mode Enable: 1 = enabled, 0 = disabled
8 to 13		
16		
30	DCE	data cache enable: 1=enabled, 0 = disabled
32	BPE	branch predictor enable: 1=enabled, 0=disabled
34	WBM	write buffer merging enable: 1 = enabled, 0 = disabled
35	SPLE	speculative load enable (1 = enable, 0 = disable) (0 default)
36		
51	D	debug mode status. this bit is set during an interrupt routine if the processor was in debug mode when the interrupt occurred.

This register supports bit set / clear CSR instructions.

DCE

Disabling the data cache is useful for some codes with large data sets to prevent cache loading of values that are used infrequently. Disabling the data cache may reduce security risks for some kinds of attacks. The instruction cache may not be disabled. Enabling / disabling the data cache is also available via the cache instruction.

BPE

Disabling branch prediction will significantly affect the cores performance but may be useful for debugging. Disabling branch prediction causes all branches to be predicted as not-taken. No entries will be updated in the branch history table if the branch predictor is disabled.

WBM bit

Merging of values stored to memory may be disabled by setting this bit. On reset write buffer merging is disabled because it is likely desirable to setup I/O devices. Many I/O devices require updates to individual bytes by separate store instructions. (Write buffer merging is not currently implemented).

SPLE

Enabling speculative loads give the processor better performance at an increased security risk to meltdown attacks.

HARTID (0x001)

This register contains a number that is externally supplied on the hartid_i input bus to represent the hardware thread id or the core number. No core should have the value zero as the hartid.

TICK (0x002)

This register contains a tick count of the number of clock cycles that have passed since the last reset. Note that this register should not be used for precise timing as the processor's clock frequency may vary for performance and power reasons. The TIME CSR may be used for wall-clock timing as it has its own timing source.

CAUSE (0x006)

This register contains a code indicating the cause of an exception or interrupt. The break handler will examine this code in order to determine what to do. Only the low order 13 bits are implemented. The high order bits read as zero and are not updateable.

STATUS (0x044)

51 37	36 29	28	27 26	25 24	23 14	13 4	3	2 1 0
~	ASID	MPRV	XS	FS	DLS	OLS	~	I_3

Bit		Meaning
0 to 2	I_3	interrupt mask
4 to 13	OLS	operating level stack (bits 4 to 5 = current operating level)
14 to 23	DLS	data level stack (bits 14 to 15 = current data level)
24 to 25	FS	floating point state
26 to 27	XS	additional core extension state
28	MPRV	memory data level swap
29 to 36	ASID	address space identifier
37 to 51		reserved

BM_CTR (0xFC1)

This register contains a 40-bit counter of the number of branch misses since the last reset.

IRQ_CTR (0xFC3)

This register is reserved to contain a 40-bit count of the number of interrupt requests.

BR_CTR (0xFC4)

This register contains a 40-bit counter of the number of branches committed since the last reset.

TIME (0xFE0, 0xFE1)

The TIME pair of registers corresponds to the wall clock real time. This register can be used to compute the current time based on a known reference point. The register value will typically be a fixed number of seconds offset from the real wall clock time. CSR 0xFE0 bits are driven by the tm_clk_i clock time base input which is independent of the cpu clock. The tm_clk_i input is a fixed frequency used for timing that cannot be less than 10MHz. The bits represent the fraction of one second. CSR 0xFE1 bits represent seconds passed. For example, if the tm_clk_i frequency is 100MHz the bits should count from 0 to 99,999,999 then cycle back to 0 again. When the bits cycle back to 0 again, the bits of the CSR 0xFE1 register is incremented.

Note that this register has a fixed time basis, unlike the TICK register whose frequency may vary with the cpu clock. The cpu clock input may vary in frequency to allow for performance and power adjustments.

TIME (0xFE1)

This register contains a reference of the number of seconds offset from the real wall clock time. It is a carry over from CSR 0xFE0 which contains the fraction of seconds value. The bits of the register represent the number of seconds passed since an arbitrary point in the past.

INSTRET (0xFE2)

This register contains a count of the number of instructions retired (successfully completed) by the core.

INFO (0xFF0 to 0xFFF)

This set of registers contains general information about the core including the manufacturer name, cpu class and name, and model number.

Memory Addressing

The cpu is word oriented with byte addressable memory. The smallest addressable unit of data is a 13-bit byte. Up to 2^{52} Bytes of data are supported and 2^{52} bytes of code.

Exceptions

All processor exceptions vector using the same vector stored in memory location $FFFFFFFFE0000. The exceptions are differentiated by the value in the exception cause register.

Cause Code	Which Exception	
0	Unimplemented instruction encountered	
1	Interrupt occurred	
2	Non-maskable interrupt occurred	
3	Processor was reset	
4	the BRK instruction was executed	

Instruction Set Summary

The instruction set includes basic arithmetic, logic, and shift instructions including: add, sub, and, or, eor, asl, rol, lsr, and ror. There are several additional instructions which are alternate mnemonics for other instructions. These include bit, and cmp.

The cycle counts are assuming no wait states are required for either instructions or data and both instructions and data can be found in the cache.

ALU Operations

ADD – Addition

Description:

Add two operand values and place the result in the target register. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

ADDIS – Add Immediate Shifted

Description:

Add an immediate value to bits 22 to 51 of register Ra_5 and place the result in target register Rt_5. Use this instruction when performing an addition with a full 52-bit constant or when building a value in a register.

Formats Supported: RIS

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

AND – Bitwise ‘And’

Description:

Bitwise ‘And’ two operand values and place the result in the target register. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

ANDIS – And Immediate Shifted

Description:

Bitwise ‘and’ an immediate value to bits 22 to 51 of register Ra_5 and place the result in target register Rt_5. Use this instruction when performing an addition with a full 52-bit constant or when building a value in a register. The constant used is one extended on the right hand side.

Formats Supported: RIS

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

ASL – Arithmetic Shift Left

Description:

Left shift one operand value by a second operand value and place the result in the target register. Zeros are shifted into the least significant bits. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

ASR – Arithmetic Shift Right

Description:

Right shift one operand value by a second operand value while preserving the sign bit and place the result in the target register. The sign bit is preserved as the shift takes place. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

BIT – Test Bits

Description:

Bitwise ‘And’ two operand values and place a result in a compare result register. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value. If the result of the ‘and’ operation is zero, a zero is stored in the compare result register. If the result of the ‘and’ operation is non-zero then either +1 or -1 is stored in the result register depending on the sign of the result.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

CMP – Comparison

Description:

Compare two operand values and store the relationship in the target compare result register. The operand values are treated as signed integers. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

If the first operand is less than the second, a minus one is stored in the target register. If the first operand equals the second a zero is stored to the target register, otherwise a positive one is stored.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

CMPU – Unsigned Comparison

Description:

Compare two operand values and store the relationship in the target compare result register. The operand values are treated as unsigned integers. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

If the first operand is less than the second, a minus one is stored in the target register. If the first operand equals the second a zero is stored to the target register, otherwise a positive one is stored.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

CSR – Control and Status Register Operation

Description:

The CSR instruction may be used to read, write or read and write a control or status register. The CSR to access is identified by a twelve-bit constant in the instruction. The current value of the CSR will be read into the register specified by the Rt field of the instruction. If Rt is zero the read data will be discarded. The CSR will be updated with the contents of the register specified by the Ra field of the instruction. If Ra is specified as register zero, then no update takes place. The CSR read / update operation is an atomic operation.

Formats Supported: CSR

Op_3		Operation
0	CSRRD	Only read the CSR, no update takes place, Rs1 should be R0.
1	CSRRW	Both read and write the CSR
2	CSRRS	Read CSR then set CSR bits
3	CSRRC	Read CSR then clear CSR bits
4		reserved
5	CSRRWI	Read and Write CSR with immediate
6	CSRRSI	Read and set using immediate
7	CSRRCI	Read and clear using immediate

CSRRS and CSRRC operations are only valid on registers that support the capability.

The OL_2 field is reserved to specify the operating level. Note that registers cannot be accessed by a lower operating level.

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

Examples:

```
CSRRD r1,r0,#CAUSECD        ; read the cause code register into r1
```

EOR – Bitwise Exclusive 'Or'

Description:

Bitwise exclusive 'Or' two operand values and place the result in the target register, updating status flags. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

LSR – Logical Shift Right

Description:

Right shift one operand value by a second operand value and place the result in the target register. Zeros are shifted into the most significant bits. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

MUL – Multiplication

Description:

Multiply two operand values and place the result in the target register. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value. Both operands are treated as signed values.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

OR – Bitwise 'Or'

Description:

Bitwise 'Or' two operand values and place the result in the target register, updating status flags. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

ORIS – Or Immediate Shifted

Description:

Bitwise 'or' an immediate value to bits 22 to 51 of register Ra_5 and place the result in target register Rt_5. Use this instruction when performing an addition with a full 52-bit constant or when building a value in a register. The constant used is zero extended on the right hand side.

Formats Supported: RIS

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

PERM – Permute Bytes

Description:

This instruction allows any combination of bytes in a source register to be copied to a target register. There are four two-bit fields which specify which source bytes to copy to the target. The source fields may be either a constant specified in the instruction, or the low order eight bits of register Rb. Field S0 indicates the source byte for target byte position 0. S1 indicates the source byte for target byte position 1. S2 and S3 work similarly for the last two target bytes. There are many interesting possibilities with this instruction. A single source byte could be copied to all target byte positions for instance. Or the order of bytes in a word could be reversed.

Formats Supported: RR, RI8

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

ROL – Rotate Left

Description:

Rotate left one operand value by a second operand value and place the result in the target register, updating status flags. The most significant bits are placed in the least significant bits. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

ROR – Rotate Right

Description:

Rotate right one operand value by a second operand value and place the result in the target register, updating status flags. The least significant bits are placed in the most significant bits. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value.

Formats Supported: RR, RI8

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

SUB – Subtraction

Description:

Subtract two operand values and place the result in the target register, updating status flags. The first operand must be in a register specified by the Ra_5 field of the instruction. The second operand may be either a register specified by the Rb_5 field of the instruction, or an immediate value. The subtraction instruction has an alternate mnemonic CMP which updates the flags differently when the target register is R0.

Formats Supported: RR, RI8, RI22, RI35

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

Memory Operations

LD – Load Data (52 bits)

Description:

Data is loaded from the memory address which is either the sum of Ra and an immediate value or the sum of Ra and Rb. Both register indirect with displacement and indexed addressing are supported.

Formats Supported: RR, RI8, RI22, RI35

Operation:

Rt = $Memory_{52}$[d+Ra]
or
Rt = $Memory_{52}$[Ra+Rb]

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

LDB – Load Data Byte (13 bits)

Description:

Data is loaded from the memory address which is the sum of Ra and an immediate value. Only register indirect with displacement addressing is supported.

Formats Supported: RI36

Flags Affected: n z

Operation:

$Rt = Memory_{13}[d+Ra]$

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

ST – Store Data (52 bits)

Description:

Data is stored to the memory address which is either the sum of Ra and an immediate value or the sum of Ra and Rb. Both register indirect with displacement and indexed addressing are supported.

Formats Supported: RR, RI8, RI22, RI35

Flags Affected: none

Operation:

$Memory_{52}[d+Ra] = Rs$
or
$Memory_{52}[Ra+Rb] = Rs$

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

STB – Store Data Byte (13 bits)

Description:

Data is stored to the memory address which is the sum of Ra and an immediate value. Only register indirect with displacement addressing is supported.

Formats Supported: RI35

Flags Affected: none

Operation:

$Memory_{13}[d+Ra] = Rs$
or
$Memory_{13}[Ra+Rb] = Rs$

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

Flow Control (Branch Unit) Operations

There are only signed versions of branches since the branch is primarily due to the output of the compare instruction which outputs values of -1,0 or +1.

BEQ – Branch if Equal

Description:

This instruction branches to the target address if the compare results register is zero. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction. This branch may be either statically or dynamically predicted.

Formats Supported: Bcc

P_2	Meaning
0	no prediction (dynamic prediction)
1	reserved
2	predict not taken (static prediction)
3	predict taken (static prediction)

Operation:

If (cr==0)
PC = NextPC + $Displacement_{12}$

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

BGE – Branch if Greater Than or Equal

Description:

This instruction branches to the target address if the compare results register is greater than or equal to zero (0 or +1). The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

Formats Supported: Bcc

Operation:

If (cr >= 0)
PC = NextPC + $Displacement_{12}$

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

BGT – Branch if Greater Than

Description:

This instruction branches to the target address if the compare results register is greater than zero (+1). The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

Formats Supported: Bcc

Operation:

If (cr > 0)
PC = NextPC + Displacement_{12}

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

BLE – Branch if Less Than or Equal

Description:

This instruction branches to the target address if the compare results register is less than or equal to zero. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

Formats Supported: Bcc

Operation:

If (cr <= 0)
PC = NextPC + $Displacement_{12}$

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

BLT – Branch if Less Than

Description:

This instruction branches to the target address if the compare results register is less than zero (-1 or -2). The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

Formats Supported: Bcc

Operation:

If (cr < 0)
PC = NextPC + $Displacement_{12}$

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

BNE – Branch if Not Equal

Description:

This instruction branches to the target address if the compare results register is not zero. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

Formats Supported: Bcc

Operation:

If (cr <> 0)
PC = NextPC + Displacement_{12}

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

BRA – Branch Always

Description:

This instruction branches unconditionally to the target address. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction. The prediction bits should be set to indicate a static prediction of taken so that the branch does not consume history table resources.

Formats Supported: Bcc

Operation:

$PC = NextPC + Displacement_{12}$

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

BRK – Break

Description:

This instruction initiates the processor break routine. The cause code register is set to four. The program counter is reset to $FFFFFFFFE0000 and instructions begin executing.

Formats Supported: BRK

Operation:

CAUSE = 4
XL = PC + 1
PC = $FFFFFFFFE0000

Execution Units: Branch

Clock Cycles:

Exceptions: none

Notes:

JMP – Jump

Description:

This instruction is an alternate mnemonic for the JAL instruction where the link register is assumed to be L0. JMP transfers execution of instructions to the address specified by the instruction. The target address may be either a 43-bit absolute address or an address contained in a register. For absolute address mode only the low order 43 bits of the program counter are affected. The upper nine bits of the program counter remain the same.

Formats Supported: ABS43, R

Flags Affected: none

Operation:

PC = $Address_{43}$
or
PC = Ra

Execution Units: Branch

Clock Cycles: 1

Exceptions: none

JAL – Jump and Link

Description:

Store the return address in the specified link register then jump to the address specified as an absolute constant or the contents of register Ra_4. The target address may be either a 43-bit absolute address or an address contained in a register. For absolute address mode only the low order 43 bits of the program counter are affected. The upper nine bits of the program counter remain the same. Note that only registers R0 to R15 may be specified as containing a jump target. This is due to limitations in the instruction encoding.

Formats Supported: ABS43, R

Flags Affected: none

Operation:

Lk = NextPC
PC = $Address_{43}$
or
PC = Ra

Execution Units: Mem

Clock Cycles: 0.5

Exceptions: none

Notes:

The next PC is either the current PC plus four when absolute addressing is used, or the current PC plus one if register indirect addressing is used.

PFI – Poll for Interrupt

Description:

The PFI instruction tests for the presence of an interrupt and performs the interrupt routine if an interrupt is present. If no interrupt is present a NOP operation is performed and the program continues with the next instruction. PFI does not check for a non-maskable (NMI) interrupt or a reset (RST). Processing for the interrupt routine begins at the universal exception handler address of $FFFFFFFFE0000.

PFI may scan three interrupt signalling lines, which lines to scan are specified by a bit-mask in the instruction.

Formats Supported: PFI

Flags Affected: none

Operation:

```
If (IRQ)
        Cause code = 1
        XL = PC + 1;
        PC = $FFFFFFFFE0000
Else
        NOP
```

Execution Units: Fetch stage

Clock Cycles:

Exceptions: none

Notes:

RST – Reset Processor

Description:

This instruction initiates the processor reset routine. The program counter is reset to $FFFFFFFFE0000 and instructions begin executing.

Formats Supported: RST

Operation:

Execution Units: Branch

Clock Cycles:

Exceptions: none

Notes:

RTI – Return from Interrupt Subroutine

Description:

Restore the previous interrupt and operating level and transfer program execution back to the address in the interrupt link register..

Formats Supported: RTI

Flags Affected: none

Operation:

OL = IOL
IM = IIM
PC = ILR

Execution Units: Mem

Clock Cycles:

Exceptions: none

Notes:

RTS – Return from Subroutine

Description:

Transfer program execution to an address stored in a link register. The link register will have been previously set by a subroutine call operation.

Formats Supported: RTS

Flags Affected: none

Operation:

PC = Lk

Execution Units: Mem

Clock Cycles: 0.5

Exceptions: none

Notes:

WAI – Wait for Interrupt

Description:

The WAI instruction waits for an interrupt to occur by holding the program counter steady. This instruction is similar to the PFI instruction except that it stops and waits for an interrupt whereas PFI doesn't wait. WAI does not check for a non-maskable (NMI) interrupt or a reset (RST).

Formats Supported: WAI

Flags Affected: none

Operation:

```
If (IRQ)
        Cause Code  = 1
        SP = SP – 4
        Memory52[SP] = PC + 1;
        SP = SP – 4
        Memory52[SP] = SR
        PC = Memory52[$FFFFFFFFFFFFC]
Else
        PC = PC
```

Execution Units: Fetch stage

Clock Cycles:

Exceptions: none

Notes:

Instruction Formats

Arithmetic / Logical

ADD							Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11 7	6 0		
		0	~3	Rb5	Ra5	Rt5	04h	ADD Rt,Ra,Rb	2
		1	Imm8		Ra5	Rt5	04h	ADD Rt,Ra,#imm8	2
	Imm22				Ra5	Rt5	14h	ADD Rt,Ra,#imm22	3
Imm35					Ra5	Rt5	24h	ADD Rt,Ra,#imm35	4

SUB							Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11 7	6 0		
		0	~3	Rb5	Ra5	Rt5	05h	Rt,Ra,Rb	2
		1	Imm8		Ra5	Rt5	05h	Rt,Ra,#imm8	2
	Imm22				Ra5	Rt5	15h	Rt,Ra,#imm22	3
Imm35					Ra5	Rt5	25h	Rt,Ra,#imm35	4

CMP								Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11	7	6 0		
		0	~3	Rb5	Ra5	~	Ct3	06h	Rt,Ra,Rb	2
		1	Imm8		Ra5	~	Ct3	06h	Rt,Ra,#imm8	2
	Imm22				Ra5	~	Ct3	16h	Rt,Ra,#imm22	3
Imm35					Ra5	~	Ct3	26h	Rt,Ra,#imm35	4

CMPU								Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11	7	6 0		
		0	~3	Rb5	Ra5	~	Ct3	07h	Rt,Ra,Rb	2
		1	Imm8		Ra5	~	Ct3	07h	Rt,Ra,#imm8	2
	Imm22				Ra5	~	Ct3	17h	Rt,Ra,#imm22	3
Imm35					Ra5	~	Ct3	27h	Rt,Ra,#imm35	4

MUL							Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0			
		0	~$_3$	Rb$_5$	Ra$_5$	Rt$_5$	0Eh	Rt,Ra,Rb	2
		1	Imm$_8$		Ra$_5$	Rt$_5$	0Eh	Rt,Ra,#imm$_8$	2
	Imm$_{22}$				Ra$_5$	Rt$_5$	1Eh	Rt,Ra,#imm$_{22}$	3
Imm$_{35}$					Ra$_5$	Rt$_5$	2Eh	Rt,Ra,#imm$_{35}$	4

AND							Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11 7	6 0		
		0	~4	Rb5	Ra5	Rt5	08h	Rt,Ra,Rb	2
		1	Imm8		Ra5	Rt5	08h	Rt,Ra,#imm6	2
	Imm22				Ra5	Rt5	18h	Rt,Ra,#imm14	3
Imm35					Ra5	Rt5	28h	Rt,Ra,#imm30	4

BIT								Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11	7	6 0		
		0	~4	Rb5	Ra5	~	Ct3	55h	Rt,Ra,Rb	2
		1	Imm8		Ra5	~	Ct3	55h	Rt,Ra,#imm6	2
	Imm22				Ra5	~	Ct3	65h	Rt,Ra,#imm14	3
Imm35					Ra5	~	Ct3	75h	Rt,Ra,#imm30	4

OR							Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11 7	6 0		
		0	~4	Rb5	Ra5	Rt5	09h	Rt,Ra,Rb	2
		1	Imm8		Ra5	Rt5	09h	Rt,Ra,#imm6	2
	Imm22				Ra5	Rt5	19h	Rt,Ra,#imm14	3
Imm35					Ra5	Rt5	29h	Rt,Ra,#imm30	4

EOR							Opcode		Bytes
51 39	38 26	25	22	21 17	16 12	11 7	6 0		
		0	~4	Rb5	Ra5	Rt5	0Ah	Rt,Ra,Rb	2
		1	Imm8		Ra5	Rt5	0Ah	Rt,Ra,#imm6	2
	Imm22				Ra5	Rt5	1Ah	Rt,Ra,#imm14	3
Imm35					Ra5	Rt5	2Ah	Rt,Ra,#imm30	4

Shifted Immediate

ADDIS					Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0		
~$_5$	Imm$_{30}$			Ra$_5$	Rt$_5$	23h	Rt,Ra,#imm$_{35}$	4

ANDIS					Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0		
~$_5$	Imm$_{30}$			Ra$_5$	Rt$_5$	22h	Rt,Ra,#imm$_{35}$	4

ORIS					Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0		
~$_5$	Imm$_{30}$			Ra$_5$	Rt$_5$	2Ch	Rt,Ra,#imm$_{35}$	4

Shift Operations

ASL							Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0			
		0	~$_3$	Rb_5	Ra_5	Rt_5	0Ch	Rt,Ra,Rb	2
		1	~$_2$	Imm_6	Ra_5	Rt_5	0Ch	Rt,Ra,#imm_6	2

ROL							Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0			
		0	~$_3$	Rb_5	Ra_5	Rt_5	1Ch	Rt,Ra,Rb	2
		1	~$_2$	Imm_6	Ra_5	Rt_5	1Ch	Rt,Ra,#imm_6	2

LSR							Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0			
		0	~$_3$	Rb_5	Ra_5	Rt_5	0Dh	Rt,Ra,Rb	2
		1	~$_2$	Imm_6	Ra_5	Rt_5	0Dh	Rt,Ra,#imm_6	2

ROR							Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0			
		0	~$_3$	Rb_5	Ra_5	Rt_5	1Dh	Rt,Ra,Rb	2
		1	~$_2$	Imm_6	Ra_5	Rt_5	1Dh	Rt,Ra,#imm_6	2

ASR							Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0			
		0	~$_3$	Rb_5	Ra_5	Rt_5	2Dh	Rt,Ra,Rb	2
		1	~$_2$	Imm_6	Ra_5	Rt_5	2Dh	Rt,Ra,#imm_6	2

Load and Store Instructions

LD						Opcode		Bytes
51 39	38 26	25	17	16 12	11 7	6 0		
		0 ~3	Rb5	Ra5	Rt5	50h	Rt.[Ra+Rb]	2
		1	Disp8	Ra5	Rt5	50h	Rt,d8[Ra]	2
	Disp22			Ra5	Rt5	60h	Rt,d22[Ra]	3
Addr35				Ra5	Rt5	70h	Rt,d35[Ra]	4

LDB						Opcode		Bytes
51 39	38 26	25	17	16 12	11 7	6 0		
		0 ~3	Rb5	Ra5	Rt5	51h	Rt.[Ra+Rb]	2
		1	Disp8	Ra5	Rt5	51h	Rt,d8[Ra]	2
	Disp22			Ra5	Rt5	61h	Rt,d22[Ra]	3
Addr35				Ra5	Rt5	71h	Rt,d35[Ra]	4

ST						Opcode		Bytes
51 39	38 26	25	17	16 12	11 7	6 0		
		0 ~3	Rb5	Ra5	Rs5	58h	Rs.[Ra+Rb]	2
		1	Disp8	Ra5	Rs5	58h	Rs,d8[Ra]	2
	Disp22			Ra5	Rs5	68h	Rs,d22[Ra]	3
Addr35				Ra5	Rs5	78h	Rs,d35[Ra]	4

STB						Opcode		Bytes
51 39	38 26	25	17	16 12	11 7	6 0		
		0 ~3	Rb5	Ra5	Rs5	59h	Rs.[Ra+Rb]	2
		1	Disp8	Ra5	Rs5	59h	Rs,d8[Ra]	2
	Disp22			Ra5	Rs5	69h	Rs,d22[Ra]	3
Addr35				Ra5	Rs5	79h	Rs,d35[Ra]	4

Flow Control

JAL	Flags:					Bytes
	Address$_{43}$		L_2	42h	JAL abs43	4
		Ra_4	L_2	48h	JAL [Ra]	1

{RTGRP}

RTS		$\sim_2$ L_2	0	44h	RTS	1
RTI		$\sim_4$	1	44h	RTI	1
RTD		$\sim_4$	2	44h	RTD	1

{WAIGRP}

PFI		$Sigmsk_4$	0	02h	PFI	1
IRQ		$\sim_4$	1	02h	IRQ	1
WAI		$Sigmsk_4$	2	02h	WAI	1
IRQ		$\sim_4$	3	02h	IRQ	1

STP		$\sim_4$	6	24o	STP	1
NOP		$\sim_4$	7	24o	NOP	1

BEQ		$Disp_{12}$	Cr_3	P_2	0	40h	BEQ disp	2
BNE		$Disp_{12}$	Cr_3	P_2	1	40h	BNE disp	2
BGT		$Disp_{12}$	Cr_3	P_2	2	40h	BGT disp	2
BLT		$Disp_{12}$	Cr_3	P_2	3	40h	BLT disp	2
BGE		$Disp_{12}$	Cr_3	P_2	0	41h	BGE disp	2
BLE		$Disp_{12}$	Cr_3	P_2	1	41h	BLE disp	2
BRA		$Disp_{12}$	Cr_3	P_2	2	41h	BRA disp	2

{BRKGRP}

RST		~4	3	00h	RST	1
NMI		~4	2	00h	NMI	1
IRQ		~4	1	00h	IRQ	1
BRK		Const4	0	00h	BRK	1

Shuffle

PERM							Opcode		Bytes
51 39	38 26	25 22	21 17	16 12	11 7	6 0			
		0	~3	Rb5	Ra5	Rt5	20h	Rt,Ra,Rb	2
		1	S3	S2	S1	S0	Ra5	Rt5	20h	Rt,Ra,#imm8	2

Control and Status Register Access

CSR	38 36	3534	33 28	27 17	16 12	11 7	6 0		
	OP3	OL2	~6	Regno12	Ra5	Rt5	01h	CSR	3

Opcode Maps

Root Level

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	BRK	CSR	{WAIGRP}		ADD	SUB	CMP	CMPU	AND	OR	EOR		ASL	LSR	MUL	
1x	{string}	{string}	{VP}		ADD	SUB	CMP	CMPU	AND	OR	EOR		ROL	ROR	MUL	
2x	PERM		ANDIS	ADDIS	ADD	SUB	CMP	CMPU	AND	OR	EOR		ORIS	ASR	MUL	
3x																
4x	{Branch}	{Branch}	JAL		{RTGRP}				JAL [R]		MTL		SEQ	SLT	FSLT	FADD
5x	LD	LDB				BIT			ST	STB	MFL		SNE	SLE	FSLE	FSUB
6x	LD	LDB				BIT			ST	STB	MTC		FSEQ	SLTU		FMUL
7x	LD	LDB				BIT			ST	STB	MFC		FSNE	SLEU		FDIV

Appendix

The first nine bits of the instruction are used as a nine-bit index into a table which holds pointers to micro-instructions.

Micro-op Instruction Format

Micro-instructions are 24 bits in size.

FL_2	$Opcode_6$	$Cnst_4$	$Src2_4$	$Src1_4$	Tgt_4	Vm_4	Z

FL_2 are two bits holding the first / last micro-instruction indicators. 00 = middle instruction, 01 = first, 10 = last, 11 = first and last.

Micro-op Fields

Tgt_4	Meaning
0	The value 0
1	Get from instruction bits 6 to 10
2	The accumulator register
3	The X register
4	The Y Register
5	The stack pointer
6	the tmp1 register
7	the tmp2 register
8	the SR register
9	the PC register (source)
10	PC + 1
11	PC + 4

$Src1_4$	Meaning
0	The value 0
1	Get from register spec instruction bits 11 to 15
2	The accumulator register
3	The X register
4	The Y Register

5	The stack pointer
6	the tmp1 register
7	the tmp2 register
8	the SR register
9	Get from register spec instruction bits 12 to 14 (Rc_3)

$Src2_4$	Meaning
0	The value 0
1	Get from register spec instruction bits 16 to 25
2	The accumulator register
3	The X register
4	The Y Register
5	The stack pointer
6	the tmp1 register
7	the tmp2 register
8	the SR register
9	the value 1
11	
15	the value -1

Vm_4	Meaning
0	scalar operation
1	vector operation use vector mask register specified in control register
8 to 15	vector operation use specified mask register from micro-op bits 1 to 3

Z	Meaning
0	merge vector results
1	zero masked vector results

$Cnst_4$	Meaning
0	the value 0
1	the value 1
2	the value 2
3	the value 3
4	the value 4
5	bits 9 to 25
6	bits 9 to 12 (branches)
7	bits 6 to 51 (JMP / JSR)
8	bits 16 to 25 of instruction
9	bits 16 to 38 of instruction
10	bits 16 to 51 of instruction
11	bits 6 to 12 (REP / SEP)
13	the value -3
14	the value -2
15	the value -1

Micro-op Lists for Instructions

BRK

```
SUB   SP,SP,#4
ST    PC+1,[SP]
SUB   SP,SP,#4
ST    FLAGS,[SP]
SEP   #i
LD    TMP,$FFFFFFFFFFFFC
JMP   0[TMP]
```

MVN

```
LD    tmp,[X]
ST    tmp,[Y]
ADD   X,X,4
ADD   Y,Y,4
SUB   AC,AC,#1
```

```
        BNE     PC

MVP
        LD      tmp,[X]
        ST      tmp,[Y]
        SUB     X,X,#4
        SUB     Y,Y,#4
        SUB     AC,AC,#1
        BNE     PC
STS
        ST      X,[Y]
        ADD     Y,Y,#4
        SUB     AC,AC,#1
        BNE     PC
CMPS
        LD      tmp1,[X]
        LD      tmp2,[Y]
        ADD     X,X,#4
        ADD     Y,Y,#4
        SUB     ac,ac,#1
        BEQ     PC+1
        CMP     tmp1,tmp2
        BEQ     PC
```

Micro-Instruction Opcodes

Opc_6	
0	ADD
1	SUB
2	CMP
3	AND
4	BIT
5	OR
6	EOR
7	LD
8	
9	LB
10	
11	ST
12	STB
13	ASL
14	ROL
15	LSR
16	ROR
17	BRA
18	BEQ
19	BNE
20	BLT
21	BGT
22	BLE
23	BGE
24	
25	
26	SEP
27	REP
28	JMP
29	STP
30	
31	