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 status flags results.

Programming Model

General Registers

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	

SR	sr	status register	conditional branching

Status Register

51 … 7	6	5	4	3	2	1	0
~	N	V	U	C	~	I	Z

Bit		Meaning
0	Z	result was zero
1	I	interrupt mask
2	~	
3	C	carry out from addition .// subtraction
4	U	user defined use
5	V	overflow
6	N	result was negative
7 to 51		reserved

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 $FFFFFFFFFFFFC. 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, 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, RI9, RI23, RI36

Flags Affected: v c n z

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, 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 AND instruction has an alternate mnemonic called BIT which updates the flags slightly differently.

Formats Supported: RR, RI9, RI23, RI36

Flags Affected: n z

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, updating status flags. 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, RI9

Flags Affected: n z

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

BIT – Bitwise ‘And’

Description:

Bitwise ‘And’ two operand values and discard the result, 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 BIT instruction is an alternate mnemonic for the AND instruction where the target register is specified as R0. The BIT instruction updates the overflow flag to the status of bit 50 of the result.

Formats Supported: RR, RI9, RI23, RI36

Flags Affected: v n z

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

CMP – Comparison

Description:

Subtract two operand values and discard the result, 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 CMP instruction is an alternate mnemonic for the SUB instruction when the target register is R0. However, unlike SUB, CMP does not update the overflow flag.

Formats Supported: RR, RI9, RI23, RI36

Flags Affected: c n z

Execution Units: ALU

Clock Cycles: 0.5

Exceptions: none

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, RI9, RI23, RI36

Flags Affected: n z

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, updating status flags. 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, RI9

Flags Affected: n z

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, RI9, RI23, RI36

Flags Affected: n z

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, RI9

Flags Affected: n z

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, RI9

Flags Affected: n z

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, RI9, RI23, RI36

Flags Affected: v c n z

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, RI9, RI23, RI36

Flags Affected: n z

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

PHP – Push Processor Status to Stack (52 bits)

Description:

The stack pointer is decremented then the status register is stored to the memory address which is contained in the stack pointer register (R31).

Formats Supported: PHP

Flags Affected: none

Operation:

```
SP = SP - 4
Memory13[SP] = SR
```

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

PLP – Pull Processor Flags from Stack (52 bits)

Description:

The status register is loaded from the memory address which is contained in the stack pointer register (R31) into a general-purpose register. Then the stack pointer is incremented.

Formats Supported: PLP

Flags Affected: c v n z i u

Operation:

Rt = Memory$_{13}$[SP]
SP = SP + 4

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

POP – Pop Data from Stack (52 bits)

Description:

Data is loaded from the memory address which is contained in the stack pointer register (R31) into a general purpose register. Then the stack pointer is incremented.

Formats Supported: POP

Flags Affected: n z

Operation:

$Rt = Memory_{13}[SP]$
$SP = SP + 4$

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

PSH – Push Data to Stack (52 bits)

Description:

The stack pointer is decremented then data from a general-purpose register is stored to the memory address which is contained in the stack pointer register (R31).

Formats Supported: PSH

Flags Affected: none

Operation:

```
SP = SP - 4
Memory$_{13}$[SP] = Rs
```

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, RI9, RI23, RI36

Flags Affected: none

Operation:

$\text{Memory}_{52}[d+Ra] = Rs$
or
$\text{Memory}_{52}[Ra+Rb] = Rs$

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: RI36

Flags Affected: none

Operation:

$\text{Memory}_{13}[\text{d+Ra}] = \text{Rs}$
or
$\text{Memory}_{13}[\text{Ra+Rb}] = \text{Rs}$

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

Exceptions: none

Flow Control (Branch Unit) Operations

JMP – Jump

Description:

Transfer execution of instructions to the address specified by the instruction. The target address may be either a 46-bit absolute address or an address contained in a register. For absolute address mode only the low order 46 bits of the program counter are affected. The upper six bits of the program counter remain the same.

Formats Supported: ABS46, R

Flags Affected: none

Operation:

$PC = Address_{46}$
or
$PC = Ra$

Execution Units: Mem

Clock Cycles: 1

Exceptions: none

JSR – Jump to Subroutine

Description:

Push the address of the next instruction on the stack, then transfer execution of instructions to the address specified by the instruction. The target address may be either a 46-bit absolute address or an address contained in a register. For absolute address mode only the low order 46 bits of the program counter are affected. The upper six bits of the program counter remain the same.

Formats Supported: ABS46, R

Flags Affected: none

Operation:

SP = SP – 4
Memory_{52}[SP] = Next PC
PC = Address_{46}
or
PC = Ra

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

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).

Formats Supported: PFI

Flags Affected: none

Operation:

```
If (IRQ)
    Cause code = 1
    SP = SP – 4
    $Memory_{52}$[SP] = PC + 1;
    SP = SP – 4
    $Memory_{52}$[SP] = SR
    PC = $Memory_{52}$[$FFFFFFFFFFFFC]
Else
    NOP
```

Execution Units: Fetch stage

Clock Cycles:

Exceptions: none

Notes:

RTI – Return from Interrupt Subroutine

Description:

Pop the status register from the stack. Next pop the address to return to from the stack, then transfer execution of instructions to that address.

Formats Supported: RTI

Flags Affected: none

Operation:

$SR = Memory_{52}[SP]$
$SP = SP + 4$
$PC = Memory_{52}[SP]$
$SP = SP + 4$

Execution Units: Mem

Clock Cycles: 8 if data is in the cache.

Exceptions: none

Notes:

RTS – Return from Subroutine

Description:

Pop the address of the next instruction from the stack, then transfer execution of instructions to that address.

Formats Supported: RTS

Flags Affected: none

Operation:

PC = Memory_{52}[SP]
SP = SP + 4

Execution Units: Mem

Clock Cycles: 4 if data is in the cache.

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
    $Memory_{52}$[SP] = PC + 1;
    SP = SP – 4
    $Memory_{52}$[SP] = SR
    PC = $Memory_{52}$[$FFFFFFFFFFFFC]
Else
    PC = PC
```

Execution Units: Fetch stage

Clock Cycles:

Exceptions: none

Notes:

Instruction Formats

Arithmetic / Logical

ADD	Flags: v c n z						Opcode		Bytes
51 39	38 26	25	21	20 16	15 11	10 6	5 0		
		0	~$_4$	Rb$_5$	Ra$_5$	Rt$_5$	00o	ADD Rt,Ra,Rb	2
		1	Imm$_9$		Ra$_5$	Rt$_5$	00o	ADD Rt,Ra,#imm$_6$	2
	Imm$_{23}$				Ra$_5$	Rt$_5$	01o	ADD Rt,Ra,#imm$_{14}$	3
Imm$_{36}$					Ra$_5$	Rt$_5$	02o	ADD Rt,Ra,#imm$_{30}$	4

SUB	Flags: v c n z						Opcode		Bytes
51 39	38 26	25	21	20 16	15 11	10 6	5 0		
		0	~$_4$	Rb$_5$	Ra$_5$	Rt$_5$	10o	ADD Rt,Ra,Rb	2
		1	Imm$_9$		Ra$_5$	Rt$_5$	10o	ADD Rt,Ra,#imm$_6$	2
	Imm$_{23}$				Ra$_5$	Rt$_5$	11o	ADD Rt,Ra,#imm$_{14}$	3
Imm$_{36}$					Ra$_5$	Rt$_5$	12o	ADD Rt,Ra,#imm$_{30}$	4

CMP	Flags: c n z						Opcode		Bytes
51 39	38 26	25	21	20 16	15 11	10 6	5 0		
		0	~$_4$	Rb$_5$	Ra$_5$	0$_5$	10o	ADD Rt,Ra,Rb	2
		1	Imm$_9$		Ra$_5$	0$_5$	10o	ADD Rt,Ra,#imm$_6$	2
	Imm$_{23}$				Ra$_5$	0$_5$	11o	ADD Rt,Ra,#imm$_{14}$	3
Imm$_{36}$					Ra$_5$	0$_5$	12o	ADD Rt,Ra,#imm$_{30}$	4

CMP is an alternate mnemonic for SUB where the target register is R0. CMP does not alter the overflow flag.

AND	Flags: n z						Opcode		Bytes
51 39	38 26	25	21	20 16	15 11	10 6	5 0		
		0	$\sim_4$	Rb_5	Ra_5	Rt_5	30o	ADD Rt,Ra,Rb	2
		1	Imm_9		Ra_5	Rt_5	30o	ADD Rt,Ra,#imm_6	2
	Imm_{23}				Ra_5	Rt_5	31o	ADD Rt,Ra,#imm_{14}	3
Imm_{36}					Ra_5	Rt_5	32o	ADD Rt,Ra,#imm_{30}	4

BIT	Flags: v n z						Opcode		Bytes
51 39	38 26	25	21	20 16	15 11	10 6	5 0		
		0	$\sim_4$	Rb_5	Ra_5	0_5	30o	ADD Rt,Ra,Rb	2
		1	Imm_9		Ra_5	0_5	30o	ADD Rt,Ra,#imm_6	2
	Imm_{23}				Ra_5	0_5	31o	ADD Rt,Ra,#imm_{14}	3
Imm_{36}					Ra_5	0_5	32o	ADD Rt,Ra,#imm_{30}	4

Bit is the AND operation with no target register; the overflow status is set to bit 50 of the result

OR	Flags: n z						Opcode		Bytes
51 39	38 26	25	21	20 16	15 11	10 6	5 0		
		0	$\sim_4$	Rb_5	Ra_5	Rt_5	40o	ADD Rt,Ra,Rb	2
		1	Imm_9		Ra_5	Rt_5	40o	ADD Rt,Ra,#imm_6	2
	Imm_{23}				Ra_5	Rt_5	41o	ADD Rt,Ra,#imm_{14}	3
Imm_{36}					Ra_5	Rt_5	42o	ADD Rt,Ra,#imm_{30}	4

EOR	Flags: n z						Opcode		Bytes
51 39	38 26	25	21	20 16	15 11	10 6	5 0		
		0	$\sim_4$	Rb_5	Ra_5	Rt_5	50o	ADD Rt,Ra,Rb	2
		1	Imm_9		Ra_5	Rt_5	50o	ADD Rt,Ra,#imm_6	2
	Imm_{23}				Ra_5	Rt_5	51o	ADD Rt,Ra,#imm_{14}	3
Imm_{36}					Ra_5	Rt_5	52o	ADD Rt,Ra,#imm_{30}	4

Shift Operations / Read-modify-write memory operations.

ASL	Flags: c n z							Opcode		Bytes
51 39		38 26	25	21	20 16	15 11	10 6	5 0		
			0	~$_4$	Rb$_5$	Ra$_5$	Rt$_5$	06o	Rt,Ra,Rb	2
			1	~$_3$	Imm$_6$	Ra$_5$	Rt$_5$	06o	Rt,Ra,#imm$_{10}$	2

ROL	Flags: c n z							Opcode		Bytes
51 39		38 26	25	21	20 16	15 11	10 6	5 0		
			0	~$_4$	Rb$_5$	Ra$_5$	Rt$_5$	16o	Rt,Ra,Rb	2
			1	~$_3$	Imm$_6$	Ra$_5$	Rt$_5$	16o	Rt,Ra,#imm$_{10}$	2

LSR	Flags: c n z							Opcode		Bytes
51 39		38 26	25	21	20 16	15 11	10 6	5 0		
			0	~$_4$	Rb$_5$	Ra$_5$	Rt$_5$	26o	Rt,Ra,Rb	2
			1	~$_3$	Imm$_6$	Ra$_5$	Rt$_5$	26o	Rt,Ra,#imm$_{10}$	2

ROR	Flags: c n z							Opcode		Bytes
51 39		38 26	25	21	20 16	15 11	10 6	5 0		
			0	~$_4$	Rb$_5$	Ra$_5$	Rt$_5$	36o	Rt,Ra,Rb	2
			1	~$_3$	Imm$_6$	Ra$_5$	Rt$_5$	36o	Rt,Ra,#imm$_{10}$	2

Load and Store Instructions

LD	Flags: n z				Opcode		Bytes
51 39	38 26	25 16	15 11	10 6	5 0		
		0 ~$_4$ Rb$_5$	Ra$_5$	Rt$_5$	60o	Rt.[Ra+Rb]	2
		1 Disp$_9$	Ra$_5$	Rt$_5$	60o	Rt,d9[Ra]	2
	Disp$_{23}$		Ra$_5$	Rt$_5$	61o	Rt,d23[Ra]	3
Addr$_{36}$			Ra$_5$	Rt$_5$	62o	Rt,d36[Ra]	4

LDB	Flags: n z				Opcode		Bytes
51 39	38 26	25 16	15 11	10 6	5 0		
Addr$_{36}$			Ra$_5$	Rt$_5$	64o	Rt,d[Ra]	4

ST	Flags: n z				Opcode		Bytes
51 39	38 26	25 16	15 11	10 6	5 0		
		0 ~$_4$ Rb$_5$	Ra$_5$	Rs$_5$	70o	Rs.[Ra+Rb]	2
		1 Disp$_9$	Ra$_5$	Rs$_5$	70o	Rs,d9[Ra]	2
	Disp$_{23}$		Ra$_5$	Rs$_5$	71o	Rs,d23[Ra]	3
Addr$_{36}$			Ra$_5$	Rs$_5$	72o	Rs,d36[Ra]	4

STB	Flags:				Opcode		Bytes
51 39	38 26	25 16	15 11	10 6	5 0		
Addr$_{36}$			Ra$_5$	Rs$_5$	74o	Ra,d[Ra]	4

Flow Control

JMP	Flags:					Bytes
	Address$_{46}$			04o	JMP abs46	4
		$\sim_2$	Ra$_5$	44o	JMP [Ra]	1

JSR	Flags:					Bytes
	Address$_{46}$			14o	JSR abs46	4
		$\sim_2$	Ra$_5$	54o	JSR [Ra]	1

RTS	Flags:	$\sim_4$	0	24o	RTS	1
RTI	Flags: z n c v b d i u	$\sim_4$	1	24o	RTI	1
PFI	Flags:	$\sim_4$	2	24o	PFI	1
IRQ	Flags:	$\sim_4$	3	24o	IRQ	1
WAI	Flags:	$\sim_4$	4	24o	WAI	1
STP	Flags:	$\sim_4$	6	24o	STP	1
NOP	Flags:	$\sim_4$	7	24o	NOP	1

BEQ	Flags:	Disp$_{17}$	0	15o	BEQ disp	2
BNE	Flags:	Disp$_{17}$	1	15o	BNE disp	2
BPL	Flags:	Disp$_{17}$	2	15o	BPL disp	2
BMI	Flags:	Disp$_{17}$	3	15o	BMI disp	2
BVS	Flags:	Disp$_{17}$	4	15o	BVS disp	2
BVC	Flags:	Disp$_{17}$	5	15o	BVC disp	2
BCS	Flags:	Disp$_{17}$	6	15o	BCS disp	2
BCC	Flags:	Disp$_{17}$	7	15o	BCC disp	2
BRA	Flags:	Disp$_{17}$	0	35o	BRA disp	2

RST	Flags: b	$\sim_4$	0	34o	RST	1
NMI	Flags: b	$\sim_4$	1	34o	NMI	1
IRQ	Flags: b	$\sim_4$	2	34o	IRQ	1
BRK	Flags: b	Const$_4$	3	34o	BRK	1

Stack push and pop operations.

PHP	Flags:	$\sim_7$		75o	PHP	1
PSH	Flags:	$\sim_2$	Ra$_5$	76o	PSH Ra	1
PLP	Flags: z c v n i b d u	$\sim_7$		65o	PLP	1
POP	Flags: n z	$\sim_2$	Rt$_5$	66o	POP Rt	1

Status Register Operations

REP	Flags: c v n z i u	Mask$_7$	56o	REP	1
SEP	Flags: c v n z i u	Mask$_7$	46o	SEP	1

String Operations

MVNB	Flags:	$\sim_4$	0	45o	MVN	1
MVPB	Flags:	$\sim_4$	1	45o	MVP	1
STSB	Flags:	$\sim_4$	2	45o	STS	1
CMPSB	Flags: c n z	$\sim_4$	3	45o	CMPS	1
MVN	Flags:	$\sim_4$	4	45o	MVN	1
MVP	Flags:	$\sim_4$	5	45o	MVP	1
STS	Flags:	$\sim_4$	6	45o	STS	1
CMPS	Flags: c n z	$\sim_4$	7	45o	CMPS	1

Opcode Maps

Root Level

	xxx000	xxx001	xxx010	xxx011	xxx100	xxx101	xxx110	xxx111
000xxx	ADD 2r,i9	ADD 2r,i23	ADD 2r,i36		JMP		ASL 2r,i6	
001xxx	SUB / CMP	SUB / CMP	SUB / CMP		JSR	Bcc d17	LSR 2r,i6	
010xxx					RTS / RTI		ROL 2r,i6	
011xxx	AND / BIT	AND / BIT	AND / BIT		BRK	Bcc d17	ROR 2r,i6	
100xxx	OR	OR	OR		JMP [Rn]	{string}	SEP	
101xxx	EOR	EOR	EOR		JSR [Rn]		REP	
110xxx	LD	LD	LD		LDB	PLP	POP	
111xxx	ST	ST	ST		STB	PHP	PSH	

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

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

$Src2_4$	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

$Cnst_4$	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,$FFFFFFFFFFFC
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		Flags Updated
0	ADD	-
1	ADD.	c v n z
2	SUB	-
3	SUB.	c v n z
4	AND.	n z
5	OR.	n z
6	EOR.	n z
7	LD	-
8	LD.	n z
9	LB	-
10	LB.	n z
11	ST	-
12	STB	-
13	ASL.	c n z
14	ROL.	c n z
15	LSR.	c n z
16	ROR.	c n z
17	BRA	-
18	BEQ	-
19	BNE	-
20	BMI	-
21	BPL	-
22	BCS	-
23	BCC	-
24	BVS	-
25	BVC	-
26	SEP	c v n z i u
27	REP	c v n z i u
28	JMP	
29	STP	
30	WAI	
31		