

SGPC Programmer's Manual

Steven Vroom

November 2016

Contents

Contents	i
List of Figures	iv
List of Tables	v
About This Manual	vi
Related Documentation	vi
Organization	vi
Conventions	vi
Acronyms and Abbreviations	vi
1 Overview	1
1.1 SGPC Architecture Overview	1
1.2 Registers	1
User-accessible Registers	1
Internal Registers	1
1.3 Instruction Conventions	1
Instruction Layout	1
Addressing Modes	1
1.4 Instruction Set	1
1.5 Interrupt Model	1
1.6 Memory Management Model	1
2 Register Set	2
2.1 Foreground Registers	2
General-Purpose Registers (GPRs)	2
Temporary Data Register	3
Stack Pointer Register (SP)	3
Program Counter Register (PC)	3
2.2 Background Registers	3
Backup Registers	3
Interrupt Registers	4
2.3 Indirect Registers	4
Flags Register	4
Segment Registers	5
2.4 Output Registers	5
3 Operand Conventions and Addressing Modes	6

3.1	Operand Conventions	6
	Bit and Byte Ordering	6
	Aligned and Misaligned Memory Access	6
3.2	Addressing Modes	6
	Not From Memory	6
	From Memory	6
4	Instruction Set Summary	7
4.1	Instruction Types	7
	Move Instructions	7
	Arithmetic Instructions	7
	Control Instructions	7
	Reserved Instructions	7
	Artifact Instructions	7
4.2	Instruction Format	7
	exceptions	7
5	Memory Management	8
5.1	Segments	8
	Base	8
	Limit	8
5.2	Code Segment (CS)	8
5.3	Data Segment (DS)	8
5.4	Segment Switching	8
5.5	Changing Segments	8
6	Interrupts	9
6.1	Interrupt Enabling/Disabling	9
	Interrupt Enabled Flag	9
	Internal Interrupt Mask	9
	Programmable Interrupt Controller (PIC)	9
6.2	State Preservation	9
	Backup	9
	Full Restore	9
	Partial Restore	9
6.3	Interrupt Service Routine (ISR)	9
	Interrupt Far Jump	9
	Return to Context	9
	Switch Context	9
7	I/O Conventions	10
7.1	Reading Input	10
7.2	Writing Output	10
	Problem with Interrupts	10
8	Instruction Set	11
8.1	MOVZ / MOV=	12
8.2	MOVNZ / MOV!=	13
8.3	MOVS	14
8.4	MOVNS	15

8.5	MOVP	16
8.6	MOVNP	17
8.7	MOV0	18
8.8	MOVNO	19
8.9	MOV0C / MOVU>	20
8.10	MOVNC / MOVU<=	21
8.11	MOVU>=	22
8.12	MOVU<	23
8.13	MOV0S>	24
8.14	MOV0S<=	25
8.15	MOV0S<	26
8.16	MOV0S>=	27
8.17	MOV	28
8.18	WRI	29
8.19	ISTR	30
8.20	ILD	31
8.21	OUT	32
8.22	IN	33
8.23	BACK	34
8.24	FRET	35
8.25	PRET	36
8.26	FJMP	37
8.27	HLT	38
8.28	NOP	39
8.29	OR	40
8.30	AND	41
8.31	XOR	42
8.32	43
A	Instruction Set Listings	44
B	Simplified Mnemonics	46
C	Common Procedures	47
D	Standard Peripherals	48
D.1	Programmable Interrupt Controller (PIC)	48
D.2	Keyboard	48
D.3	Programmable Interrupt Timer (PIT)	48
D.4	Sound Card	48
D.5	Graphical Card	48
D.6	Memory Control Hub (MCH)	48
D.7	Segments and Out Of Bounds Exception	48

List of Figures

List of Tables

2.1	List of Foreground Registers	2
2.2	List of Background Registers	3
2.3	List of Flags	4
2.4	List of Segment Registers	5
2.5	List of Output Registers	5
A.1	List of instructions sorted by opcode	44
A.1	List of instructions sorted by opcode	45

About This Manual

Related Documentation

Organization

Conventions

Acronyms and Abbreviations

Chapter 1

Overview

1.1 SGPC Architecture Overview

1.2 Registers

User-accessible Registers

Internal Registers

1.3 Instruction Conventions

Instruction Layout

Addressing Modes

1.4 Instruction Set

1.5 Interrupt Model

1.6 Memory Management Model

Chapter 2

Register Set

This chapter describes the registers separated in four groups based on accessibility. However, the internal registers are omitted from this chapter since these are implementation specific.

2.1 Foreground Registers

The foreground registers are the registers all regular instructions can read from and write to. There are eight 8-bit and eight 16-bit foreground registers. These registers are preserved in interrupts.

Table 2.1: List of Foreground Registers

ID	Mnemonic	Descriptive Name	Length in bits
0x0	al	The lower byte of ax	8
0x1	ah	The higher byte of ax	8
0x2	bl	The lower byte of bx	8
0x3	bh	The higher byte of bx	8
0x4	cl	The lower byte of cx	8
0x5	ch	The higher byte of cx	8
0x6	dl	The lower byte of dx	8
0x7	dh	The higher byte of dx	8
0x8	ax	The first GPR	16
0x9	bx	The second GPR	16
0xA	cx	The third GPR	16
0xB	dx	The fourth GPR	16
0xC	ex	The fifth GPR	16
0xD	tm	Temporary data	16
0xE	sp	Stack pointer	16
0xF	pc	Program counter	16

General-Purpose Registers (GPRs)

These registers are meant for computing storage. The first four 16-bit registers are all splitted into two 8-bit registers. So software can directly access the upper and lower byte of these 16-bit registers.

Temporary Data Register

This registers is meant to facilitate call procedures. So it won't be preserved in a function call. However this nothing more than a suggestion to the user, software can use this register as a regular GPR.

Stack Pointer Register (SP)

This register is meant to keep track of the end of the stack. However this nothing more than a suggestion to the user, software can use this register as a regular GPR.

Program Counter Register (PC)

This register holds the address of the next instruction to run. Writing to this registers means jumping to other code.

2.2 Background Registers

Background registers can only accessed with the instructions BSTR and BLD.

Table 2.2: List of Background Registers

ID	Mnemonic	Descriptive Name	Length in bits
0x0	n/a	Reserved	8
0x1	n/a	Reserved	8
0x2	n/a	Reserved	8
0x3	n/a	Reserved	8
0x4	n/a	Reserved	8
0x5	n/a	Reserved	8
0x6	n/a	Reserved	8
0x7	n/a	Reserved	8
0x8	bpc	Program counter backup	16
0x9	bsf	Segments and flags backup	16
0xA	ipc	Interrupt program counter	16
0xB	is	Interrupt segments	16
0xC	n/a	Reserved	16
0xD	n/a	Reserved	16
0xE	n/a	Reserved	16
0xF	n/a	Reserved	16

Backup Registers

The backup registers are used to backup the state of the CPU (see section 6.2). The foreground registers, segment registers and flags register all have their own backup register. However, most backup register aren't background registers. Only the segment registers, program counter register and flags register have directly accessible backup registers. Note that both segment registers and the flag register share one 16-bit backup register.

Interrupt Registers

The Interrupt registers hold the new state the CPU should jump to when an interrupt is triggered (see chapter 6). Only the segment registers and the program counter have an interrupt register.

2.3 Indirect Registers

Indirect registers are registers that software can't directly read nor write to with any instruction. All the backup registers that aren't background registers fall under this category. All the indirect registers can only be written to and read from via the state preservation system (see section 6.2).

Flags Register

The flags register holds multiple flags (see Table 2.3). There are two types of flags: status flags and control flags. Status flags give software extra information about the last computation made, while control flags control how the cpu behaves. There is only one control flag in the [insert name here] processor, the interrupts enabled flag. If this flag is set to zero interrupt requests will be ignored (see section 6.1).

Zero flag : If the result of the last computation is equal to zero this flag will be set, otherwise it will be cleared.

Sign flag : If the most significant bit of the result of the last computation is set this flag will be set, otherwise it will be cleared.

Parity flag : If the least significant bit of the result of the last computation is clear this flag will be set, otherwise it will be cleared.

Overflow flag : If the signed two's-complement result of the last computation is too large to fit in operand A (see section 3.1) this flag will be set, otherwise it will be cleared. (Not all computational instructions change this flag)

Carry flag : If the unsigned result of the last computation is too large to fit in operand A (see section 3.1) this flag will be set, otherwise it will be cleared. (Not all computational instructions change this flag)

Table 2.3: List of Flags

Mnemonic	Descriptive Name	Type of flag	Length in bits
Z	Zero flag	Status flag	1
S	Sign flag	Status flag	1
P	Parity flag	Status flag	1
O	Overflow flag	Status flag	1
C	Carry flag	Status flag	1
I	Interrupts enabled flag	Control flag	1

Segment Registers

The segment registers (see Table 2.4) hold the index of the segments currently used (see chapter 5).

Table 2.4: List of Segment Registers

Mnemonic	Descriptive Name	Length in bits
CS	Code segment index	5
DS	Data segment index	5

2.4 Output Registers

The output registers (see Table 2.5) hold information that is sent to the peripherals (see chapter 7)

Table 2.5: List of Output Registers

Mnemonic	Descriptive Name	Length in bits
AO	The first output register	16
BO	The second output register	16
CO	The third output register	16
DO	The fourth output register	16
EO	The fifth output register	16
FO	The sixth output register	16
GO	The seventh output register	16
HO	The eighth output register	16

Chapter 3

Operand Conventions and Addressing Modes

This chapter describes

3.1 Operand Conventions

Bit and Byte Ordering

Aligned and Misaligned Memory Access

3.2 Addressing Modes

Not From Memory

Register Direct

Absolute

Register with displacement

From Memory

Direct

Base Plus Displacement

Chapter 4

Instruction Set Summary

This chapter gives a summary on the instruction set of the [insert name here] processor. It will describe the types of instructions and the binary layout of the instructions.

4.1 Instruction Types

The instructions can be separated in five types. Two of these types are not meant to be used.

Move Instructions

Arithimic Instructions

Control Instructions

Reserved Instructions

Artifact Instructions

Artifact Instructions are instructions with defined behaviour, but aren't considered a permanent part of the [insert name here] processor family. These instructions tend to

4.2 Instruction Format

exceptions

Chapter 5

Memory Management

5.1 Segments

Base

Limit

5.2 Code Segment (CS)

5.3 Data Segment (DS)

5.4 Segment Switching

5.5 Changing Segments

Chapter 6

Interrupts

6.1 Interrupt Enabling/Disabling

Interrupt Enabled Flag

Internal Interrupt Mask

Programmable Interrupt Controller (PIC)

6.2 State Preservation

Backup

Full Restore

Partial Restore

6.3 Interrupt Service Routine (ISR)

Interrupt Far Jump

Return to Context

Switch Context

Chapter 7

I/O Conventions

7.1 Reading Input

7.2 Writing Output

Problem with Interrupts

Chapter 8

Instruction Set

8.1 MOVZ / MOV=

Move if zero / Move if equal

Usages

0x00: MOVZ a b

0x00: MOV= a b

Description

Writes B to A if the zero flag is set. This instruction can also be used to only write B to A if, in the last comparison, B was equal to A. The alternative Memnonic MOV= was provided for this use. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

if Z **then**

$a \leftarrow b$

end if

alternative pseudocode:

if $b_{previous} = a_{previous}$ **then**
 operands of a previous calculation

$a \leftarrow b$

end if

▷ Where $b_{previous}$ and $a_{previous}$ are the

Flags Affected

none

8.2 MOVNZ / MOV!=

Move if not zero / Move if not equal

Usages

0x01: MOVNZ a b

0x01: MOV!= a b

Description

Writes B to A if the zero flag is clear. This instruction can also be used to only write B to A if, in the last comparison, B was unequal to A. The alternative Memmonic MOV= was provided for this use. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

if $\neg Z$ **then**

$a \leftarrow b$

end if

alternative pseudocode:

if $b_{previous} \neq a_{previous}$ **then**
 operands of a previous calculation

$a \leftarrow b$

end if

▷ Where $b_{previous}$ and $a_{previous}$ are the

Flags Affected

none

8.3 MOVS

Move if sign

Usages

0x02: MOVS a b

Description

Writes B to A if the sign flag is set.

Operation

```
if  $S$  then  
   $a \leftarrow b$   
end if
```

Flags Affected

none

8.4 MOVNS

Move if not sign

Usages

0x03: MOVNS a b

Description

Writes B to A if the sign flag is clear.

Operation

```
if  $\neg S$  then  
   $a \leftarrow b$   
end if
```

Flags Affected

none

8.5 MOVP

Move if parity

Usages

0x04: MOVP a b

Description

Writes B to A if the parity flag is set.

Operation

```
if  $P$  then
   $a \leftarrow b$ 
end if
```

Flags Affected

none

8.6 MOVNP

Move if not parity

Usages

0x05: MOVNP a b

Description

Writes B to A if the parity flag is clear.

Operation

```
if  $\neg P$  then  
   $a \leftarrow b$   
end if
```

Flags Affected

none

8.7 MOVO

Move if overflow

Usages

0x06: MOVO a b

Description

Writes B to A if the overflow flag is set.

Operation

```
if O then
   $a \leftarrow b$ 
end if
```

Flags Affected

none

8.8 MOVNO

Move if no overflow

Usages

0x07: MOVNO a b

Description

Writes B to A if the overflow flag is clear.

Operation

```
if  $\neg O$  then  
   $a \leftarrow b$   
end if
```

Flags Affected

none

8.9 MOVC / MOVU>

Move if carry / Move if unsigned greater

Usages

0x08: MOVC a b

0x08: MOVU> a b

Description

Writes B to A if the carry flag is set. This instruction can also be used to only write B to A if, in the last comparison of unsigned numbers, B was greater than A. The alternative Memnonic MOVU> was provided for this use. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

if C **then**

$a \leftarrow b$

end if

alternative pseudocode:

if $b_{previous} > a_{previous}$ **then**
 operands of a previous calculation

$a \leftarrow b$

end if

▷ Where $b_{previous}$ and $a_{previous}$ are the

Flags Affected

none

8.10 MOVNC / MOVU<=**Move if no carry / Move if unsigned smaller or equal****Usages**

0x09: MOVNC a b

0x09: MOVU<= a b

Description

Writes B to A if the carry flag is clear. This instruction can also be used to only write B to A if, in the last comparison of unsigned numbers, B was smaller than or equal to A. The alternative Memmonic MOVU<= was provided for this use. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation**if $\neg C$ then** $a \leftarrow b$ **end if**

alternative pseudocode:

if $b_{previous} \leq a_{previous}$ then \triangleright Where $b_{previous}$ and $a_{previous}$ are the
 operands of a previous calculation

 $a \leftarrow b$ **end if****Flags Affected**

none

8.11 MOVU \geq

Move if unsigned greater or equal

Usages

0x0A: MOVU \geq a b

Description

Writes B to A if, in the last comparison of unsigned numbers, B was greater than or equal to A. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

```

if  $C \vee Z$  then
     $a \leftarrow b$ 
end if

```

alternative pseudocode:

```

if  $b_{previous} \geq a_{previous}$  then           ▷ Where  $b_{previous}$  and  $a_{previous}$  are the
    operands of a previous calculation
     $a \leftarrow b$ 
end if

```

Flags Affected

none

8.12 MOVU<

Move if unsigned smaller

Usages

0x0B: MOVU< a b

Description

Writes B to A if, in the last comparison of unsigned numbers, B was smaller than A. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

```
if  $\neg(C \vee Z)$  then  
     $a \leftarrow b$   
end if
```

alternative pseudocode:

```
if  $b_{previous} < a_{previous}$  then  
     $a \leftarrow b$   
end if
```

▷ Where $b_{previous}$ and $a_{previous}$ are the operands of a previous calculation

Flags Affected

none

8.13 MOVS>

Move if signed greater

Usages

0x0C: MOVS> a b

Description

Writes B to A if, in the last comparison of signed numbers, B was greater than A. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

```
if  $\neg Z \wedge (O \oplus S)$  then  
     $a \leftarrow b$   
end if
```

alternative pseudocode:

```
if  $b_{previous} > a_{previous}$  then  
     $a \leftarrow b$   
end if
```

▷ Where $b_{previous}$ and $a_{previous}$ are the operands of a previous calculation

Flags Affected

none

8.14 MOVS<=**Move if signed smaller or equal****Usages**

0x0D: MOVS<= a b

Description

Writes B to A if, in the last comparison of signed numbers, B was smaller than or equal to A. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

```

if  $Z \vee \neg(O \oplus S)$  then
   $a \leftarrow b$ 
end if

```

alternative pseudocode:

```

if  $b_{previous} \leq a_{previous}$  then           ▷ Where  $b_{previous}$  and  $a_{previous}$  are the
  operands of a previous calculation
   $a \leftarrow b$ 
end if

```

Flags Affected

none

8.15 MOVS<

Move if signed smaller

Usages

0x0E: MOVS< a b

Description

Writes B to A if, in the last comparison of signed numbers, B was smaller than A. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

```

if  $O \oplus S$  then
   $a \leftarrow b$ 
end if

```

alternative pseudocode:

```

if  $b_{previous} < a_{previous}$  then           ▷ Where  $b_{previous}$  and  $a_{previous}$  are the
  operands of a previous calculation
   $a \leftarrow b$ 
end if

```

Flags Affected

none

8.16 MOVS \geq **Move if signed greater or equal****Usages**

0x0F: MOVS \geq a b

Description

Writes B to A if, in the last comparison of signed numbers, B was greater than or equal to A. The comparison use of this instruction can only be expected to work if the last calculation was a compare or subtract instruction, and the used status flags haven't changed in the meanwhile.

Operation

```

if  $\neg O \oplus S$  then
   $a \leftarrow b$ 
end if

```

alternative pseudocode:

```

if  $b_{previous} \geq a_{previous}$  then           ▷ Where  $b_{previous}$  and  $a_{previous}$  are the
  operands of a previous calculation
   $a \leftarrow b$ 
end if

```

Flags Affected

none

8.17 MOV

Move

Usages

0x10: MOV a b

Description

Writes B to A unconditionally.

Operation

$$a \leftarrow b$$

Flags Affected

none

8.18 WRI

Write interrupts enabled

Usages

0x12: WRI _ b

Description

Writes the least significant bit of B to the interrupts enabled flag. This allows you to enable and disable interrupts in a single instruction.

Operation

$I \leftarrow b \& 0001_{16}$ \triangleright Everything but the least significant bit is omitted

Flags Affected

I

8.19 ISTR

Internal store

Usages

0x14: ISTR a b

Description

Writes B to internal register A.

Operation

$$a \leftarrow b$$

▷ Where a is an internal register

Flags Affected

none

8.20 ILD

Internal load

Usages

0x15: ILD a b

Description

Writes internal register B to A.

Operation

$$a \leftarrow b$$

▷ Where b is an internal register

Flags Affected

none

8.21 OUT

Output

Usages

0x16: OUT a b

Description

Writes B to output register A.

Operation

$$a \leftarrow b$$

▷ Where a is an output register

Flags Affected

none

8.22 IN

Input

Usages

0x17: IN a b

Description

Request input from pheriperal B and write the returned input to A.

Operation

$$a \leftarrow \text{REQUESTINPUTFROM}(b)$$

Flags Affected

none

8.23 BACK

Backup

Usages

0x18: BACK __ __

Description

Backups all the registers that have a backup registers

Operation

$baa \leftarrow aa$

$bbx \leftarrow bx$

$bcx \leftarrow cx$

$bdx \leftarrow dx$

$bea \leftarrow ea$

$btm \leftarrow tm$

$bsp \leftarrow sp$

$bpc \leftarrow pc$

$bsf \leftarrow sf$

▷ Backup the segment and flag registers

Flags Affected

none

8.24 FRET

Full restore

Usages

0x19: FRET __ __

Description

Restores some of the registers and marks that there is no interrupt being handled

Operation

$ax \leftarrow bax$

$bx \leftarrow bbx$

$cx \leftarrow bcx$

$dx \leftarrow bdx$

$ex \leftarrow bex$

$tm \leftarrow btm$

$sp \leftarrow bsp$

$pc \leftarrow bpc$

$sf \leftarrow bsf$

MARKINTERRUPTENDED()
interrupts

▷ Restore the segment and flag registers

▷ Unlock the internal lock that prevents

Flags Affected

Z, S, P, O, C and I

8.25 PRET

Partial restore

Usages

0x1A: PRET __ __

Description

Restores some of the registers and marks that the there is no interrupt being handled.

Operation

$pc \leftarrow bpc$	
$sf \leftarrow bsf$	▷ Restore the segment and flag registers
MARKINTERRUPTENDED()	▷ Unlock the internal lock that prevents
interrupts	

Flags Affected

Z, S, P, O, C and I

8.26 FJMP

Far jump

Usages

0x1B: FJMP __ __

Description

Restores some of the registers.

Operation

$$pc \leftarrow bpc$$
$$sf \leftarrow bsf$$

▷ Restore the segment and flag registers

Flags Affected

Z, S, P, O, C and I

8.27 HLT

Halt

Usages

0x1C: HLT __ __

Description

Halts the CPU until an external interrupt is triggered and resumes.

Operation

HALTCPU()

Flags Affected

none

8.28 NOP

No operations

Usages

0x1D: NOP __ __

Description

Does nothing but delay the cpu a minimal amount of cycles.

Operation

```
if false then           ▷ NOP is implemented as a do never conditional
...
end if
```

Flags Affected

none

8.29 OR

Bitwise or

Usages

0x20: OR a b
0x21: OR !a b
0x22: OR a !b
0x23: OR !a !b

Description

performs bitwise logical or operation on A and B and writes the answer to A. The answer is also evaluated to update the status flags. Both operands can be One's Complement Negated (NOT'ed) before the operation in the same instruction. This is marked by the '!' before the operand.

Operation

$$y \leftarrow (!)a \mid (!)b$$
$$a \leftarrow y$$
$$Z, S, P \leftarrow \text{EVALUATEANSWER}(y)$$

Flags Affected

Z, S and P

8.30 AND

Bitwise and

Usages

0x24: AND a b
0x25: AND !a b
0x26: AND a !b
0x27: AND !a !b

Description

performs bitwise logical and operation on A and B and writes the answer to A. The answer is also evaluated to update the status flags. Both operands can be One's Complement Negated (NOT'ed) before the operation in the same instruction. This is marked by the '!' before the operand.

Operation

$$y \leftarrow (!)a \ \& \ (!)b$$
$$a \leftarrow y$$
$$Z, S, P \leftarrow \text{EVALUATEANSWER}(y)$$

Flags Affected

Z, S and P

8.31 XOR

Bitwise exclusive or

Usages

0x28: XOR a b

0x29: XOR !a b

0x2A: XOR a !b (*Artifact instruction*)

0x2B: XOR !a !b (*Artifact instruction*)

Description

performs bitwise logical exclusive or operation on A and B and writes the answer to A. The answer is also evaluated to update the status flags. Both operands can be One's Complement Negated (NOT'ed) before the operation in the same instruction. This is marked by the '!' before the operand. However because of the logical properties of exclusive or, "XOR !a !b" and "XOR a !b" behave exactly the same as "XOR a b" and "XOR !a b" respectively. Thus instruction "XOR !a !b" and "XOR a !b" are deemed to be artifact instructions and shouldn't be used.

Operation

$$y \leftarrow (!)a \wedge (!)b$$

$$a \leftarrow y$$

$$Z, S, P \leftarrow \text{EVALUATEANSWER}(y)$$

Flags Affected

Z, S and P

8.32**Usages**

0x: a b

Description**Operation**

$$a \leftarrow b$$

Flags Affected

none

Appendix A

Instruction Set Listings

Table A.1: List of instructions sorted by opcode

Opcode			Memmonic	Operand A	Operand B
Decimal	Hex	Binary			
0	0x00	000000	MOVZ & MOV=	W?	R?
1	0x01	000001	MOVNZ & MOV!=	W?	R?
2	0x02	000010	MOVS	W?	R?
3	0x03	000011	MOVNS	W?	R?
4	0x04	000100	MOV P	W?	R?
5	0x05	000101	MOVNP	W?	R?
6	0x06	000110	MOV O	W?	R?
7	0x07	000111	MOVNO	W?	R?
8	0x08	001000	MOV C & MOVU>	W?	R?
9	0x09	001001	MOVNC & MOVU<=	W?	R?
10	0x0A	001010	MOVU>=	W?	R?
11	0x0B	001011	MOVU<	W?	R?
12	0x0C	001100	MOV S>	W?	R?
13	0x0D	001101	MOV S<=	W?	R?
14	0x0E	001110	MOV S<	W?	R?
15	0x0F	001111	MOV S>=	W?	R?
16	0x10	010000	MOV	W	R
17	0x11	010001	n/a	n/a	n/a
18	0x12	010010	WRI	—	R
19	0x13	010011	n/a	n/a	n/a
20	0x14	010100	STRB	O	R
21	0x15	010101	LDB	W	O
22	0x16	010110	OUT	O	R
23	0x17	010111	IN	W	O
24	0x18	011000	BACK	—	—
25	0x19	011001	FRET	—	—
26	0x1A	011010	PRET	—	—
27	0x1B	011011	FJMP	—	—
28	0x1C	011100	HLT	—	—
29	0x1D	011101	NOP	—	—
30	0x1E	011110	CMP	R	R

Table A.1: List of instructions sorted by opcode

Opcode			Memmonic	Operand A	Operand B
Decimal	Hex	Binary			
31	0x1F	011111	TEST	R	R
32	0x20	100000	OR	R&W	R
33	0x21	100001	OR	!R&W	R
34	0x22	100010	OR	R&W	!R
35	0x23	100011	OR	!R&W	!R
36	0x24	100100	AND	R&W	R
37	0x25	100101	AND	!R&W	R
38	0x26	100110	AND	R&W	!R
39	0x27	100111	AND	!R&W	!R
40	0x28	101000	XOR	R&W	R
41	0x29	101001	XOR	!R&W	R
42	0x2A	101010	XOR	R&W	!R
43	0x2B	101011	XOR	!R&W	!R
44	0x2C	101100	ADD	R&W	R
45	0x2D	101101	ADD	!R&W	R
46	0x2E	101110	ADD	R&W	!R
47	0x2F	101111	ADD	!R&W	!R
48	0x30	110000	ADD1	R&W	R
49	0x31	110001	ADD1	!R&W	R
50	0x32	110010	ADD1	R&W	!R
51	0x33	110011	ADD1	!R&W	!R
52	0x34	110100	ADDC	R&W	R
53	0x35	110101	ADDC	!R&W	R
54	0x36	110110	ADDC	R&W	!R
55	0x37	110111	ADDC	!R&W	!R
56	0x38	111000	SHL	W	R
57	0x39	111001	SHL1	W	R
58	0x3A	111010	RCL	W	R
59	0x3B	111011	ROL	W	R
60	0x3C	111100	SHR	W	R
61	0x3D	111101	SHR1	W	R
62	0x3E	111110	RCR	W	R
63	0x3F	111111	ROR	W	R

Appendix B

Simplified Mnemonics

Appendix C

Common Procedures

Appendix D

Standard Peripherals

D.1 Programmable Interrupt Controller (PIC)

D.2 Keyboard

D.3 Programmable Interrupt Timer (PIT)

D.4 Sound Card

D.5 Graphical Card

D.6 Memory Control Hub (MCH)

D.7 Segements and Out Of Bounds Exception