

SGPC Programmer's Manual

Steven Vroom

November 2016

Contents

| | |
|--|------------|
| Contents | i |
| List of Figures | v |
| List of Tables | vi |
| About This Manual | vii |
| Related Documentation | vii |
| Organization | vii |
| Conventions | vii |
| Acronyms and Abbreviations | vii |
| 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 | 6 |

| | | |
|----------|---|-----------|
| 3.1 | Operands in Instructions | 6 |
| 3.2 | Normal Operands | 6 |
| | Exception: ISTR & ILD | 6 |
| 3.3 | Special Operands | 6 |
| | Special Operand A | 7 |
| | Special Operand B | 7 |
| 3.4 | List of Operands | 7 |
| 4 | Addressing | 10 |
| 4.1 | Bit and Byte Ordering | 10 |
| 4.2 | Aligned and Misaligned Memory Access | 10 |
| 5 | Instruction Set Summary | 11 |
| 5.1 | Instruction Types | 11 |
| | Move Instructions | 11 |
| | Addition and Subtraction Instructions | 12 |
| | Logical Instructions | 12 |
| | Bit Shifting Instructions | 12 |
| | Control Instructions | 13 |
| | Reserved Instructions | 13 |
| | Artifact Instructions | 14 |
| 5.2 | Instruction Format | 14 |
| 6 | Memory Management | 15 |
| 6.1 | Segments | 15 |
| | Base | 15 |
| | Limit | 15 |
| 6.2 | Code Segment (CS) | 15 |
| 6.3 | Data Segment (DS) | 15 |
| 6.4 | Segment Switching | 15 |
| 6.5 | Changing Segments | 15 |
| 7 | Interrupts | 16 |
| 7.1 | Interrupt Enabling/Disabling | 16 |
| | Interrupt Enabled Flag | 16 |
| | Internal Interrupt Mask | 16 |
| | Programmable Interrupt Controller (PIC) | 16 |
| 7.2 | State Preservation | 16 |
| | Backup | 16 |
| | Full Restore | 16 |
| | Partial Restore | 16 |
| 7.3 | Interrupt Service Routine (ISR) | 16 |
| | Interrupt Far Jump | 16 |
| | Return to Context | 16 |
| | Switch Context | 16 |
| 8 | I/O Conventions | 17 |
| 8.1 | Reading Input | 17 |
| 8.2 | Writing Output | 17 |
| | Problem with Interrupts | 17 |

| | | |
|----------|---|-----------|
| 9 | Instruction Set | 18 |
| 9.1 | MOVZ / MOV= | 19 |
| 9.2 | MOVNZ / MOV!= | 20 |
| 9.3 | MOVS | 21 |
| 9.4 | MOVNS | 22 |
| 9.5 | MOVP | 23 |
| 9.6 | MOVNP | 24 |
| 9.7 | MOV0 | 25 |
| 9.8 | MOVNO | 26 |
| 9.9 | MOV0 / MOVU> | 27 |
| 9.10 | MOVNC / MOVU<= | 28 |
| 9.11 | MOVU>= | 29 |
| 9.12 | MOVU< | 30 |
| 9.13 | MOVS> | 31 |
| 9.14 | MOVS<= | 32 |
| 9.15 | MOVS< | 33 |
| 9.16 | MOVS>= | 34 |
| 9.17 | MOV | 35 |
| 9.18 | WRI | 36 |
| 9.19 | ISTR | 37 |
| 9.20 | ILD | 38 |
| 9.21 | OUT | 39 |
| 9.22 | IN | 40 |
| 9.23 | BACK | 41 |
| 9.24 | FRET | 42 |
| 9.25 | PRET | 43 |
| 9.26 | FJMP | 44 |
| 9.27 | HLT | 45 |
| 9.28 | NOP | 46 |
| 9.29 | OR | 47 |
| 9.30 | AND | 48 |
| 9.31 | XOR | 49 |
| 9.32 | ADD / SUB | 50 |
| 9.33 | ADD1 / SUB1 | 51 |
| 9.34 | ADDC / SUBB | 52 |
| 9.35 | | 53 |
| A | Instruction Set Listings | 54 |
| B | Simplified Mnemonics | 56 |
| C | Common Procedures | 57 |
| D | Standard Peripherals | 58 |
| D.1 | Programmable Interrupt Controller (PIC) | 58 |
| D.2 | Keyboard | 58 |
| D.3 | Programmable Interrupt Timer (PIT) | 58 |
| D.4 | Sound Card | 58 |
| D.5 | Graphical Card | 58 |
| D.6 | Memory Control Hub (MCH) | 58 |

CONTENTS

iv

| | |
|---|----|
| D.7 Segements and Out Of Bounds Exception | 58 |
|---|----|

List of Figures

| | | |
|-----|------------------------------|----|
| 2.1 | Instruction Layout | 4 |
| 5.1 | Instruction Layout | 14 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | List of Foreground Registers | 2 |
| 2.2 | List of Background Registers | 3 |
| 2.3 | List of Flags | 5 |
| 2.4 | List of Segment Registers | 5 |
| 2.5 | List of Output Registers | 5 |
| 3.1 | List of Operands | 7 |
| 3.1 | List of Operands | 8 |
| 3.1 | List of Operands | 9 |
| A.1 | List of instructions sorted by opcode | 54 |
| A.1 | List of instructions sorted by opcode | 55 |

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

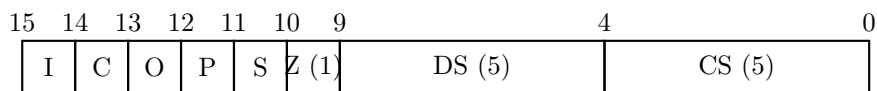
The backup registers are used to backup the state of the CPU (see section 7.2). The foreground registers, segment registers and flags register all have their own backup register. However, most backup register aren't background registers. Only the segment registers, program counter register and flags register have directly accessible backup registers. Note that both segment registers and the flag register share one 16-bit backup register.

CS : The code segment, This holds the index of the segment that is used to fetch instructions from the memory (see ??).

DS : The data segment, This holds the index of the segment that is used to read data from and write data to the memory (see ??).

Z, S, P, O, C and I : The flags, these hold data about the state of the processor or control how the processor behaves (see section 2.3)

Figure 2.1: Instruction Layout



Interrupt Registers

The Interrupt registers hold the new state the CPU should jump to when an interrupt is triggered (see chapter 7). 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 7.2).

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 7.1).

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 ??) 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 ??) 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 6).

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 8)

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

This chapter describes the possible operands and how to use them. The operands are separated into normal operands and special operands.

3.1 Operands in Instructions

All instructions have two operands, however, some instructions don't use them or only use one. The two operands have a set role. Operand A (the first operand after the instruction) is the output and the secondary input. Operand B (the second operand after the instruction) is the primary input. In other words the processor generally follows this behaviour: read B, optionally read A, calculate and finally write to A.

3.2 Normal Operands

Normal operands simply access the foreground registers. This type of operands is ensured to take the least amount of read and write time, so it is advised to use this operand type for computations.

Exception: ISTR & ILD

The ISTR and ILD alter the behaviour of the normal A and B operands respectively. Instead of the foreground registers the operands access the background registers. There are only four background registers. If ISTR or ILD try to access any of the reserved background register the behaviour of that instruction is undefined.

3.3 Special Operands

Special operands make use of the immediate of an instruction. If both operands of an instruction are special operands then they share the same immediate. The A and B operand have different special operands.

Special Operand A

Special operand A is used to access the memory. The operands adds a register to the immediate and uses the answer of that calculation as memory address. It's also possible to use the immediate as address directly without the addition. It's possible to read and write both 8-bits and 16-bit.

Special Operand B

Special operand B can both be used access the memory or directly. The operands either adds a register to the immediate and uses the answer of that calculation as memory address, or just uses the answer of the calculation directly as the value of the operand. It's also possible to use the immediate as address or value directly without the addition.

3.4 List of Operands

Table 3.1: List of Operands

| ID | Mnemonic | Description | Length in bits |
|---|----------|---------------------------|----------------|
| Normal Operands | | | |
| 0x00 | al | The lower byte of ax | 8 |
| 0x01 | ah | The higher byte of ax | 8 |
| 0x02 | bl | The lower byte of bx | 8 |
| 0x03 | bh | The higher byte of bx | 8 |
| 0x04 | cl | The lower byte of cx | 8 |
| 0x05 | ch | The higher byte of cx | 8 |
| 0x06 | dl | The lower byte of dx | 8 |
| 0x07 | dh | The higher byte of dx | 8 |
| 0x08 | ax | The first GPR | 16 |
| 0x09 | bx | The second GPR | 16 |
| 0x0A | cx | The third GPR | 16 |
| 0x0B | dx | The fourth GPR | 16 |
| 0x0C | ex | The fifth GPR | 16 |
| 0x0D | tm | Temporary data | 16 |
| 0x0E | sp | Stack pointer | 16 |
| 0x0F | pc | Program counter | 16 |
| Exceptions: Operand A of ISTR & Operand B of ILD | | | |
| 0x00 | n/a | Reserved | 8 |
| 0x01 | n/a | Reserved | 8 |
| 0x02 | n/a | Reserved | 8 |
| 0x03 | n/a | Reserved | 8 |
| 0x04 | n/a | Reserved | 8 |
| 0x05 | n/a | Reserved | 8 |
| 0x06 | n/a | Reserved | 8 |
| 0x07 | n/a | Reserved | 8 |
| 0x08 | bpc | Program counter backup | 16 |
| 0x09 | bsf | Segments and flags backup | 16 |
| 0x0A | ipc | Interrupt program counter | 16 |
| 0x0B | is | Interrupt segments | 16 |

Table 3.1: List of Operands

| ID | Mnemonic | Description | Length in bits |
|------------------------------------|----------|--------------------------------------|----------------|
| 0x0C | n/a | Reserved | 16 |
| 0x0D | n/a | Reserved | 16 |
| 0x0E | n/a | Reserved | 16 |
| 0x0F | n/a | Reserved | 16 |
| Exception: Operand A of OUT | | | |
| 0x00 | AO | The first output register | 16 |
| 0x01 | BO | The second output register | 16 |
| 0x02 | CO | The third output register | 16 |
| 0x03 | DO | The fourth output register | 16 |
| 0x04 | EO | The fifth output register | 16 |
| 0x05 | FO | The sixth output register | 16 |
| 0x06 | GO | The seventh output register | 16 |
| 0x07 | HO | The eighth output register | 16 |
| 0x08 | n/a | Reserved | 16 |
| 0x09 | n/a | Reserved | 16 |
| 0x0A | n/a | Reserved | 16 |
| 0x0B | n/a | Reserved | 16 |
| 0x0C | n/a | Reserved | 16 |
| 0x0D | n/a | Reserved | 16 |
| 0x0E | n/a | Reserved | 16 |
| 0x0F | n/a | Reserved | 16 |
| Special Operands A | | | |
| 0x10 | [ax+#] | The byte at: ax plus the immediate | 8 |
| 0x11 | [bx+#] | The byte at: bx plus the immediate | 8 |
| 0x12 | [cx+#] | The byte at: cx plus the immediate | 8 |
| 0x13 | [dx+#] | The byte at: dx plus the immediate | 8 |
| 0x14 | [fx+#] | The byte at: fx plus the immediate | 8 |
| 0x15 | [pc+#] | The byte at: pc plus the immediate | 8 |
| 0x16 | [sp+#] | The byte at: sp plus the immediate | 8 |
| 0x17 | [#] | The byte at: the immediate | 8 |
| 0x18 | @[ax+#] | The word at: ax plus the immediate | 16 |
| 0x19 | @[bx+#] | The word at: bx plus the immediate | 16 |
| 0x1A | @[cx+#] | The word at: cx plus the immediate | 16 |
| 0x1B | @[dx+#] | The word at: dx plus the immediate | 16 |
| 0x1C | @[fx+#] | The word at: fx plus the immediate | 16 |
| 0x1D | @[pc+#] | The word at: pc plus the immediate | 16 |
| 0x1E | @[sp+#] | The word at: sp plus the immediate | 16 |
| 0x1F | @[#] | The word at: the immediate | 16 |
| Special Operands B | | | |
| 0x10 | [ax+#] | The memory at: ax plus the immediate | (same as A) |
| 0x11 | [bx+#] | The memory at: bx plus the immediate | (same as A) |
| 0x12 | [cx+#] | The memory at: cx plus the immediate | (same as A) |
| 0x13 | [dx+#] | The memory at: dx plus the immediate | (same as A) |
| 0x14 | [fx+#] | The memory at: fx plus the immediate | (same as A) |
| 0x15 | [pc+#] | The memory at: pc plus the immediate | (same as A) |
| 0x16 | [sp+#] | The memory at: sp plus the immediate | (same as A) |
| 0x17 | [#] | The memory at: the immediate | (same as A) |

Table 3.1: List of Operands

| ID | Mnemonic | Description | Length in bits |
|------|----------|-----------------------|----------------|
| 0x18 | ax+# | ax plus the immediate | 16 |
| 0x19 | bx+# | bx plus the immediate | 16 |
| 0x1A | cx+# | cx plus the immediate | 16 |
| 0x1B | dx+# | dx plus the immediate | 16 |
| 0x1C | fx+# | fx plus the immediate | 16 |
| 0x1D | pc+# | pc plus the immediate | 16 |
| 0x1E | sp+# | sp plus the immediate | 16 |
| 0x1F | # | the immediate | 16 |

Chapter 4

Addressing

4.1 Bit and Byte Ordering

4.2 Aligned and Misaligned Memory Access

Chapter 5

Instruction Set Summary

This chapter gives a summary on the instruction set of the ABCD processor. It will describe the types of instructions and the binary layout of the instructions.

5.1 Instruction Types

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

Move Instructions

The move instructions copy data from its source operand to its target operand. The simplest move instruction is MOV and does just that. The rest of the move instructions are conditional move instructions, these only copy data when certain conditions are met. These conditions all have to do with previous calculations and use the flags register.

MOVZ : Move if the zero flag is set.

MOVNZ : Move if the zero flag is clear.

MOVS : Move if the sign flag is set.

MOVNS : Move if the sign flag is clear.

MOVP : Move if the parity flag is set.

MOVNP : Move if the parity flag is clear.

MOVZ : Move if the overflow flag is set.

MOVNO : Move if the overflow flag is clear.

MOVC : Move if the carry flag is set.

MOVNC : Move if the carry flag is clear.

MOV= : Move if the previous comparison was equal.

MOV!= : Move if the previous comparison was unequal.

MOVU< : Move if the previous comparison of unsigned integers was smaller.

MOVU<= : Move if the previous comparison of unsigned integers was smaller or equal.

MOVU>= : Move if the previous comparison of unsigned integers was greater or equal.

MOVU> : Move if the previous comparison of unsigned integers was greater.

MOVS< : Move if the previous comparison of signed integers was smaller.

MOVS<= : Move if the previous comparison of signed integers was smaller or equal.

MOVS>= : Move if the previous comparison of signed integers was greater or equal.

MOVS> : Move if the previous comparison of signed integers was greater.

Addition and Subtraction Instructions

There preform binary addition and subtraction for both signed and unsigned integers. In all the addition instructions you can two's compliment negate one of the operands. A subtraction instruction is an addition instruction with the B operand negated. There are three groups of addition/subtraction instructions:

ADD / SUB : Add/subtract the two operands.

ADD1 / SUB1 : Add/subtract the two operands and add/subtract 1.

ADDC / SUBB : Add/subtract the two operands and, if the carry flag is set, add/subtract 1.

Logical Instructions

The logical instructions preform simple bitwise logical functions. In all logical Instructions you can one's compliment negate (NOT) one or both of the operands.

OR : Perform bitwise logical OR.

AND : Perform bitwise logical AND.

XOR : Perform bitwise logical XOR.

Bit Shifting Instructions

The bit shifting instructions shift or rotate the bits of its operand by one bit.

SHL : Shifts all bits one bit left and clears the new least significant bit.

SHR : Shifts all bits one bit right and clears the new most significant bit.

SHL1 : Shifts all bits one bit left and sets the new least significant bit.

SHR1 : Shifts all bits one bit right and sets the new most significant bit.

ROL : Shifts all bits one bit left and writes the old most significant bit to the new least significant bit.

ROR : Shifts all bits one bit right and writes the old least significant bit to the new most significant bit

RCL : Shifts all bits one bit left and clears new the least significant bit.

RCR : Shifts all bits one bit right and clears new the most significant bit.

Control Instructions

Control Instruction change specific parts of the processor state. Unlike the move and computational instructions these instructions don't necessarily read from operand B and write/read from A .

WRI : Sets or clears the I flag.

ISTR : Writes to an background register.

ILD : Writes from an background register.

OUT : Writes to an output register.

IN : Request input from a pheriperal.

BACK : Backup the most registers.

FRET : Fully restore the last backup and marks and marks any running interrupt done.

PRET : Restore the program counter and the stack pointer from the last backup and marks any running interrupt done.

FJMP : Restore the program counter and the stack pointer from the last backup.

HLT : Halts the processor untill a interrupt is triggered.

NOP : Does nothing but delay the processor a bit.

Reserved Instructions

Reserved instructions are instructions with undefined behaviour, meaning they can do anything to the state of the CPU or the RAM. Reserved instructions should never be used in code. However, reserved instructions will never message pheripers, so it's impossible to change non-volatile memory or damage any components. Thus any problems that reserved instructions invoke directly will not persist after restarting the system.

Artifact Instructions

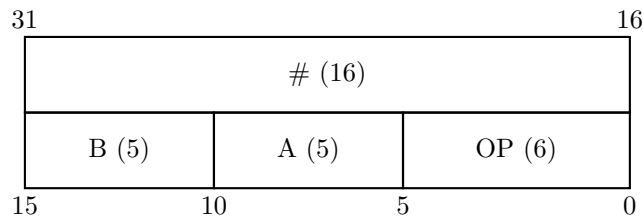
Artifact instructions are instructions with defined behaviour, but aren't considered a permanent part of the ABCD processor family. These instructions tend to only exist because it would take more parts to remove them than it takes to leave them in.

5.2 Instruction Format

All instructions follow one single format. The format has four parts:

- OP** : The opcode, this describes what operation will be executed by this instruction (see Appendix A).
- A** : Operand A, This describes what the instruction will use as output and secondary input (see Table 3.1).
- B** : Operand B, This describes what the instruction will use primary input (see Table 3.1).
- #** : The immediate, This is number is used by some operands to add to it's data or offset it's address.

Figure 5.1: Instruction Layout



Chapter 6

Memory Management

6.1 Segments

Base

Limit

6.2 Code Segment (CS)

6.3 Data Segment (DS)

6.4 Segment Switching

6.5 Changing Segments

Chapter 7

Interrupts

7.1 Interrupt Enabling/Disabling

Interrupt Enabled Flag

Internal Interrupt Mask

Programmable Interrupt Controller (PIC)

7.2 State Preservation

Backup

Full Restore

Partial Restore

7.3 Interrupt Service Routine (ISR)

Interrupt Far Jump

Return to Context

Switch Context

Chapter 8

I/O Conventions

8.1 Reading Input

8.2 Writing Output

Problem with Interrupts

Chapter 9

Instruction Set

9.1 MOVZ / MOV=

Move if zero / Move if equal

Usages

0x00: MOVZ a b

0x00: MOV= a b (*Alternative notation*)

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

9.2 MOVNZ / MOV!=

Move if not zero / Move if not equal

Usages

0x01: MOVNZ a b

0x01: MOV!= a b (*Alternative notation*)

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 ¬Z then
  a ← b
end if
```

alternative pseudocode:

```
if  $b_{previous} \neq a_{previous}$  then
  a ← b
end if
```

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

Flags Affected

none

9.3 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

9.4 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

9.5 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

9.6 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

9.7 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

9.8 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

9.9 MOVC / MOVU>

Move if carry / Move if unsigned greater

Usages

0x08: MOVC a b

0x08: MOVU> a b (*Alternative notation*)

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

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

0x09: MOVNC a b

0x09: MOVU<= a b (*Alternative notation*)**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 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 $\neg C$ 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

9.11 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

9.12 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

9.13 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

9.14 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

9.15 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  
   $a \leftarrow b$   
end if
```

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

Flags Affected

none

9.16 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

9.17 MOV

Move

Usages

0x10: MOV a b

Description

Writes B to A unconditionally.

Operation

$$a \leftarrow b$$

Flags Affected

none

9.18 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

9.19 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

9.20 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

9.21 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

9.22 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

9.23 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

9.24 FRET

Full restore

Usages

0x19: FRET __ __

Description

Restores some of the registers and marks that the 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

9.25 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

9.26 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

9.27 HLT

Halt

Usages

0x1C: HLT __ __

Description

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

Operation

HALTCPU()

Flags Affected

none

9.28 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

9.29 OR

Bitwise or

Usages

0x20: OR a b
0x21: OR !a b
0x22: OR a !b
0x23: OR !a !b
0x23: NAND a b (*Alternative notation*)

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

9.30 AND

Bitwise and

Usages

0x24: AND a b

0x25: AND !a b

0x26: AND a !b

0x27: AND !a !b

0x27: NOR a b (*Alternative notation*)

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

9.31 XOR

Bitwise exclusive or

Usages

0x28: XOR a b
 0x29: XOR !a b
 0x29: XAND a b (*Alternative notation*)
 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

9.32 ADD / SUB

Addition / Subtraction

Usages

0x28: ADD a b

0x29: ADD -a b

0x2A: ADD a -b

0x2A: SUB a b (*Alternative notation*)

Description

adds A to B and writes the answer to A. The addition is checked on overflow and carry and the answer is also evaluated to update the status flags. Both operands can be Two's Complement Negated before the operation in the same instruction. This is marked by the '-' before the operand.

Operation

$$y \leftarrow (-)a + (-)b$$

$$a \leftarrow y$$

$$O, C \leftarrow \text{CHECKADDITION}()$$

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

Flags Affected

Z, S, P, O and C

9.33 ADD1 / SUB1

Addition plus 1 / Subtraction minus 1

Usages

0x28: ADD1 a b (*Artifact instruction*)
0x29: ADD1 -a b (*Artifact instruction*)
0x2A: ADD1 a -b (*Artifact instruction*)
0x2A: SUB1 a b (*Artifact instruction*) (*Alternative notation*)

Description

adds A and 1 to B and writes the answer to A. The addition is checked on overflow and carry and the answer is also evaluated to update the status flags. Both operands can be Two's Complement Negated before the operation in the same instruction. This is marked by the '-' before the operand. If one of the operands is negated the extra 1 will be negated as well

Operation

$$y \leftarrow (-)a + (-)b + (-)1$$
$$a \leftarrow y$$
$$O, C \leftarrow \text{CHECKADDITION}()$$
$$Z, S, P \leftarrow \text{EVALUATEANSWER}(y)$$

Flags Affected

Z, S, P, O and C

9.34 ADDC / SUBB

Addition with carry / Subtraction with borrow

Usages

0x28: ADDC a b

0x29: ADDC -a b

0x2A: ADDC a -b

0x2A: SUBB a b (*Alternative notation*)

Description

adds A to B and writes the answer to A, however, if the carry flag is set an extra 1 will be added to the sum. The addition is checked on overflow and carry and the answer is also evaluated to update the status flags. Both operands can be Two's Complement Negated before the operation in the same instruction. This is marked by the '-' before the operand. If one of the operands is negated the extra 1 will be negated as well.

Operation

$$y \leftarrow (-)a + (-)b + (-)C$$

$$a \leftarrow y$$

$$O, C \leftarrow \text{CHECKADDITION}()$$

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

Flags Affected

Z, S, P, O and C

9.35**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 & SUB | R&W | !R |
| 47 | 0x2F | 101111 | n/a | n/a | n/a |
| 48 | 0x30 | 110000 | ADD1 | R&W | R |
| 49 | 0x31 | 110001 | ADD1 | !R&W | R |
| 50 | 0x32 | 110010 | ADD1 & SUB1 | R&W | !R |
| 51 | 0x33 | 110011 | n/a | n/a | n/a |
| 52 | 0x34 | 110100 | ADDC | R&W | R |
| 53 | 0x35 | 110101 | ADDC | !R&W | R |
| 54 | 0x36 | 110110 | ADDC & SUBB | R&W | !R |
| 55 | 0x37 | 110111 | n/a | n/a | n/a |
| 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