SRP16 Assembler Guide

v0.1.0

Vishnu Shankar B

Contents

Ch 1. Assembler Preprocessors

1.1. Comments

Anything that begins with ';' - semicolon are comments. Comments are ignored.

1.2. Labels

Anything that ends with ':' character is a label.

1.3. `.byte` Preprocessor

Lets you define a byte.

Example:

```
.byte 0x08
```

1.4. `.hex` Preprocessor

Lets you define array of bytes.

Example:

```
.hex "AABBCC"
```

1.5. `.string` Preprocessor

Lets of define an array of character bytes.

Example:

```
.string "Hello world"
.byte 0x00 ;Null character
```

1.6. `.equ` Preprocessor

`.equ` lets you define constants.

Example:

```
.equ "zero", 0 ;"zero" is now 0
```

1.7. .org Preprocessor

`.org` Lets you align data or instructions to a particular address.

Example:

```
.org 0x08
.byte 0x01 ;This byte will be at address 0x08
```

1.8. `.include` Preprocessor

`.include` Lets you include code from other files.

Example:

```
.include "code.asm" ;Copy paste code from code.asm
```

Ch 2. SRP16 Instruction Set Summary

2.1. Registers

- General Purpose Registers R0-R15 (Accessible by Load-Store Instructions)
- General Purpose Registers R16-R31 (Not Accessible by Load-Store Instructions)
- Accumulator Register (R60)
- Memory Pointer Register or MPTR (R61)
- Stack Pointer or SP (R62)
- Program Counter or PC (R63)
- POP, PUSH, INC, DEC instructions can only access General Purpose Registers R0-R31

2.2. Instruction Set

Instruction	Operation
LDR Rx, 8-bit-signed-immediate	Rx ← immediate
LDRU Rx, 8-bit-unsigned-immediate	Rx[15:8] ← immediate
LD@MPTR Rx, 8-bit-signed-offset	Rx ← memory[MPTR] MPTR ← MPTR+offset
ST@MPTR Rx, 8-bit-signed-offset	memory[MPTR] ← Rx MPTR ← MPTR+offset
LDB@MPTR Rx, 8-bit-signed-offset	Rx[7:0] ← memory[MPTR] MPTR ← MPTR+offset
STB@MPTR Rx, 8-bit-signed-offset	memory[MPTR] ← Rx[7:0] MPTR ← MPTR+offset
LDA 12-bit-signed-immediate	A ← immediate
LDAU 6-bit-unsigned-immediate	A[15:12] ← immediate[3:0]
LDMPTR 12-bit-unsigned-immediate	MPTR ← immediate
LDMPTRU 12-bit-signed-immediate	MPTR[15:12] ← immediate[3:0]
MOV Rx, Ry	Rx ← Ry
MOV Rx, PC	Rx ← PC+4
JMP Ry or MOV PC, Ry	PC ← Ry
SJMP 12-bit-signed-offset	PC ← PC+offset
SJMPF 12-bit-signed-offset	if(flag): PC ← PC+offset
NOTF	flag ← !flag
POP Rx	Rx ← memory[SP] SP ← SP+1
PUSH Rx	SP ← SP-1 memory[SP] ← Rx
INC Rx	Rx ← Rx+1
DEC Rx	Rx ← Rx-1

Instruction	Operation
ADDI 8-bit-signed-immediate	A ← A+immediate
ADCI 8-bit-signed-immediate	A ← A+immediate+carry
SBBI 8-bit-signed-immediate	A ← A-immediate-carry
ANDI 8-bit-signed-immediate	A ← A&immediate
ORI 8-bit-signed-immediate	A ← A
XORI 8-bit-signed-immediate	A ← A^immediate
SLAI 6-bit-unsigned-immediate	A ← A<<<immediate
SRAI 6-bit-unsigned-immediate	A ← A>>>immediate
SLLI 6-bit-unsigned-immediate	A ← A<<immediate
SRLI 6-bit-unsigned-immediate	A ← A>>immediate
ADD Rx	A ← A+Rx
SUB Rx	A ← A-Rx
ADC Rx	A ← A+Rx+carry
SBB Rx	A ← A-Rx-carry
AND Rx	A ← A&Rx
OR Rx	A ← A
XOR Rx	A ← A^Rx
SLA Rx	A ← A<<<Rx
SRA Rx	A ← A>>>Rx
SLL Rx	A ← A<<Rx
SRL Rx	A ← A>>Rx
CLI 8-bit-signed-immediate	if(A<immediate): flag ← 1 else: flag ← 0
CGI 8-bit-signed-immediate	if(A>immediate): flag ← 1 else: flag ← 0
CEI 8-bit-signed-immediate	if(A==immediate): flag ← 1 else: flag ← 0
CL Rx	if(A<Rx): flag ← 1 else: flag ← 0
CG Rx	if(A>Rx): flag ← 1 else: flag ← 0
CE Rx	if(A==Rx): flag ← 1 else: flag ← 0