CYPRESS

CY16
USB Host/Slave Controller/16-Bit RISC Processor Programmers Guide

Version 1.1

Cypress Semiconductor
3901 North First Street
San Jose, CA 95134
Tel.: (800) 858-1810 (toll-free in the U.S.)
(408) 943-2600
www.cypress.com

CY16 USB Host/Slave Controller/16-Bit RISC Processor Programmers Guide Version 1.1.

Table of Contents

Chapter 1 CPU Instruction Formats and Hardware Specific Details

1.1 Introduction

This document describes the assembly language programming environment for the CY16 Instruction Set, Registers and Addressing modes, etc.. A complete description of all the assembler instructions is provided.

1.2 General

The CY16 processor uses a unified program and data memory space; although this RAM is also integrated into the CY16 core, provisions have been made for external memory as well.

The CY16 RISC processor incorporates 2 sets of 16 CPU registers (selected with a REGBANK register) along with a Flags Register, Interrupt Enable, and many other control registers.

It is important to remember the CY16 is a byte addressable processor, which supports byte moves and even-aligned word moves. A simplified functional block diagram of the heart of the processor is shown in Figure 1-1.

The model in Figure 1-1 will help the programmer understand the effects of byte and word operations from an assembly syntax point-of-view. This is a usage model only and does not reflect the actual hardware architecture.

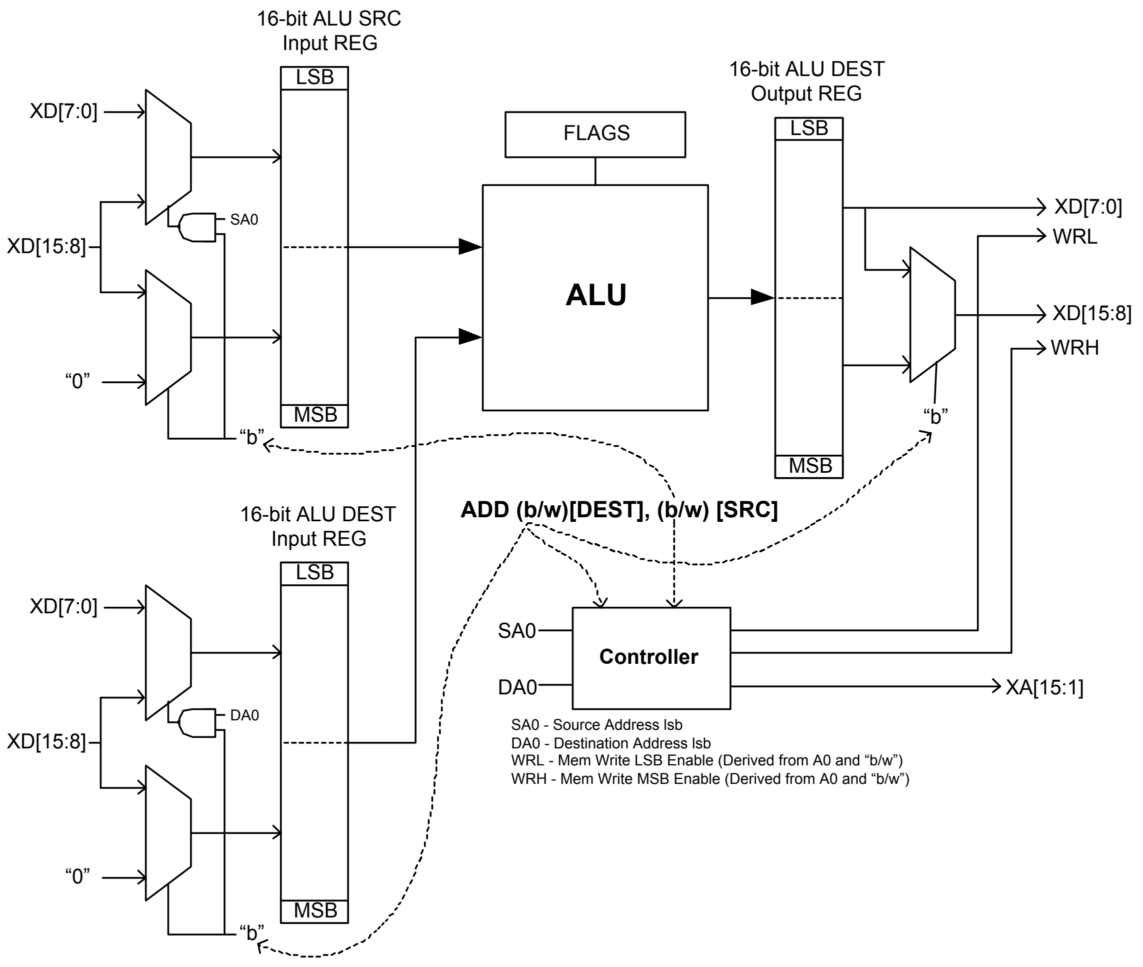


Figure 1-1. Simplified Functional Block Diagram

1.3 Register Set

The CY16 Processor incorporates 16-bit general-purpose registers called R0..R15, a REGBANK register, a program counter, along with various other registers. The function of each register is defined as follows:

Table 1-1. Register Name and Function

Name	Function
General Purpose Registers	R0-R7
General Purpose/Address Registers	R8-R14
REGBANK	Forms base address for registers R0-R15
Flags	Contains Flags (defined below)
Program Counter	PC
Stack Pointer	R15
Interrupt Enable (0xC00E)	Bit masks to enable/disable various interrupts

1.3.1 General Purpose Registers

The general-purpose registers can be used to store intermediate results, and to pass parameters to and return them from subroutine calls.

1.3.2 General Purpose/Address Registers

In addition to acting as general-purpose registers, registers R8-R14 can also serve as pointer registers. Instructions can access RAM locations by referring to any of these registers. In normal operation, register R15 is reserved for use as a stack pointer.

1.3.3 REGBANK Register (0xC002: R/W)

Registers R0..R15 are mapped into RAM via the REGBANK register. The REGBANK register is loaded with a base address, of which the 11 most significant bits are used. A read from or a write to one of the registers will generate a RAM address by:

- Shifting the 4 least significant bits of the register number left by 1.
- OR-ing the shifted bits of the register number with the upper 11 bits of the REGBANK register.
- Forcing the Least Significant Bit to 0.

For example, if the REGBANK register is left at its default value of 100 hex, a read of register R14 would read address 11C hex.

Register	Hex Value	Binary Value															
REGBANK	0100	0	0	0	0	0	0	0	1	0	0	0	x	x	x	x	x
R14	000E << 1 = 001C	x	x	x	x	x	x	x	x	x	x	0	1	1	1	0	0
RAM Location	011C	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0

Note: *Regardless of the value loaded into the REGBANK register, bits 0..4 will be ignored.*

1.3.4 Flags Register (0xC000: R/W)

The CY16 Processor uses the flags listed below.

FLAG																
Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	I	S	O	C	Z

Flag	Definition
Z	**Zero**: Instruction execution resulted in a result of 0.
C	**Carry/Borrow**: Arithmetic instruction resulted in a carry (for addition) or a borrow (for subtraction).
O	**Overflow**: Arithmetic result was either larger than the destination operand size (for addition) or smaller than the destination operand should allow for subtraction.
S	**Sign**: Set if MS result bit is “1” .
I	**Global Interrupts**: Enabled if “1”.

Notes:

1. FLAGS ARE SET FOR 16-BIT OPERATIONS ONLY!
2. Flag behavior for each instruction will be described in the following sections.
3. The FLAG Register should be Pushed and Popped to the Stack for hardware ISRs.

1.3.5 Program Counter

The Program Counter is an internal 16-bit register. The contents of this register will be pushed onto the stack following either an interrupt or a call instruction and popped from the stack following a return instruction.

1.3.6 Reset Vector

On receiving a hardware reset, the CY16 Processor jumps to address 0xFFF0, which is an internal ROM address.

1.3.7 Hardware Interrupt Servicing

The CY16 has 48 hardware interrupt vectors. Each interrupt has a special purpose as described in the Hardware Technical Reference Manual.

When a hardware interrupt occurs, program execution jumps to the interrupt vector's address and global interrupts are disabled (i.e. the CY16 does an implied CLI). The rest of the haardware interrupt service must be supplied by the programmer, i.e. push/pop flags, STI and RET.

A template for a hardware ISR is shown below:

```
INT_XX_ISR:
   push [0xc000]          ; push flags
   ;-- push registers --
   ;
   ;== USER CODE ==
   ;
   ;-- pop registers --
   pop [0xc000]           ; pop flags
   sti
   ret
```

This template must be used for all hardware interrupt service routines. Software Interrupts are effectively the same as CALLs, so these operations are not required.

1.3.8 General Instruction Format

To understand addressing modes supported by the CY16 Processor, you must know how the instruction format is defined. In general, the instructions include four bits for the instruction ***opcode***, six bits for the source operand, and six bits for the destination operand.

ADD																
Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Opcode				Source						Destination					

Some instructions, especially single operand-operator and program control instructions, will not adhere strictly to this format. They will be discussed in detail in the following sections.

1.3.9 Addressing Modes

This section describes in detail the six-operand field bits referred to in the previous section as ***source*** and ***destination***. Bear in mind that although the discussion refers to bits 0 through 5, the same bit definitions apply to the "source" operand field, bits 6 through 11. These are the basic addressing modes in the CY16 Processor.

Table 1-2. Addressing Modes

Mode	**5**	**4**	**3**	**2**	**1**	**0**
Register	0	0	r	r	r	r
Immediate	0	1	1	1	1	1
Direct	1	0	b/w	1	1	1
Indirect	0	1	b/w	r	r	r
Indirect with Auto Increment	1	0	b/w	r	r	r
Indirect with Index	1	1	b/w	r	r	r

Notes:

1. b/w: '1' for byte-wide access, '0' for word access.
2. Immediate mode not valid as a destination.
3. Indirect with auto increment and byte-wide Indirect addressing is illegal with the stack pointer (R15).

1.3.9.1 Byte vs Word Addressing

The CY16 Processor supports byte and word data access for direct and indirect source and destination operands. However, this is a memory access modifier only. I.e. all internal operations are 16 bit. Fetching or storing a byte operand always uses the low byte of the 16-bit internal ALU registers. It is not possible to fetch a byte from memory and place it in the high byte of a register or visa-versa. Also, the setting of flags are based on internal 16-bit register operations. Hence there is no support for signed byte arithmetic and all shift and rotate operations are 16 bits etc.

1.3.10 Register Addressing

In Register Addressing, any one of registers R0-R15 can be selected using bits 0-3. If register addressing is used, operands are always 16-bit operands, since all registers are 16-bit registers. For example, an instruction using register R7 as an operand would fill the operand field like this:

Bits	**5**	**4**	**3**	**2**	**1**	**0**
Register Operand	0	0	0	1	1	1

1.3.11 Immediate Addressing

In Immediate Addressing, the instruction word is immediately followed by the source operand. For example, the operand field would be filled with:

Bits	**5**	**4**	**3**	**2**	**1**	**0**
Operand field	0	1	1	1	1	1

Note: *In immediate addressing, the source operand must be 16 bits wide, eliminating the need for a b/w bit.*

1.3.12 Direct Addressing

In Direct Addressing, the word following the instruction word is used as an address into RAM. Again, the operand can be either byte or word sized, depending on the state of bit 3 of the operand field. For example, to do a word-wide read from a direct address, the ***source*** operand field would be formed like this:

Bits	**5**	**4**	**3**	**2**	**1**	**0**
I/O operand	1	0	0	1	1	1

Note: *For a memory-to-memory move, the instruction word would be followed by two words, the first being the* ***source*** *address and the second being the* ***destination****.*

1.3.13 Indirect Addressing

Indirect addressing is accomplished using address registers R8-15. In indirect addressing, the operand is found at the memory address pointed to by the register. Since only eight address registers exist, only three bits are required to select an address register. For example, register R10 (binary 1010) can be selected by ignoring bit 3, leaving the bits 010. Bit 3 of the operand field is then used as the byte/word bit, set to “0” to select word or “1” to select byte addressing. In this example, a byte-wide operand is selected at the memory location pointed to by register R10:

Bits	**5**	**4**	**3**	**2**	**1**	**0**
Memory operand	0	1	1	0	1	0

Note: *For register R15, byte-wide operands are prohibited. If bit 3 is set high, the instruction is decoded differently.*

1.3.14 Indirect Addressing with Auto Increment

Indirect Addressing with Auto Increment works identically to Indirect Addressing, except that at the end of the read or write cycle, the register is incremented by 1 or 2 (depending whether it is a byte-wide or word-wide access.)

This mode is prohibited for register R15. If bits 0..2 are all high, the instruction is decoded differently.

1.3.15 Indirect Addressing with Offset

In Indirect Addressing with Offset, the instruction word is followed by a 16-bit word that is added to the contents of the address register to form the address for the operand. The offset is an unsigned 16-bit word, and will “wrap” to low memory addresses if the register and offset add up to a value greater than the size of the processor’s address space.

1.3.16 Stack Pointer (R15) Special Handling

Register R15 is designated as the Stack Pointer, and has these special behaviors:

- If addressed in indirect mode, the register pre-decrements on a write instruction, and post-increments on a read instruction, emulating Push and Pop instructions.
- Byte-wide reads or writes are prohibited in indirect mode.
- If R15 is addressed in Indirect with Index mode, it does not auto-increment or auto-decrement.

1.3.17 SW Call Stack Details

The CY16 initializes R15 to the top of the software Call Stack. When data is pushed on to the stack the stack pointer (R15) is decremented and the value is moved to the memory location pointed to by R15. Hence, the stack grows toward smaller addresses and R15 always points to the last item on the stack. For a subroutine CALL or software INT, the return address is pushed onto the stack and the PC is set to the branch address for the instruction. An RET instruction pops the return address off the stack and sets the PC to that address.

Chapter 2 CY16 CPU Instruction Set

2.1 General

The instruction set can be roughly divided into three classes of instructions:

- **Dual Operand Instructions** (Instructions with two operands – a source and a destination)
- **Program Control Instructions** (Jump, Call and Return)
- **Single Operand Instructions** (Instructions with only one operand – a destination)

2.2 Detailed Instruction Formats

2.2.1 Dual Operand Instructions

MOV																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0000				Source						Destination					

Destination := Source
Flags Affected: None

ADD																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0001				Source						Destination					

Destination := Destination + Source
Flags Affected: Z, C, O, S

ADDC																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0010				Source						Destination					

Destination := Destination + Source + Carry Flag
Flags Affected: Z, C, O, S

SUB																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0011				Source						Destination					

Destination := Destination - Source
Flags Affected: Z, C, O, S

SUBB																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0100				Source						Destination					

Destination := Destination - Source - Carry Flag
Flags Affected: Z, C, O, S

CMP																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0101				Source						Destination					

[Not saved] := Destination - Source
Flags Affected: Z, C, O, S

AND																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0110				Source						Destination					

Destination := Destination & Source
Flags Affected: Z, S

TEST																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0111				Source						Destination					

[Not saved] := Destination & Source
Flags Affected: Z, S

OR																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1000				Source						Destination					

Destination := Destination | Source
Flags Affected: Z, S

XOR																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1001				Source						Destination					

Destination := Destination ^ Source
Flags Affected: Z, S

2.2.2 Program Control Instructions

Jcc JUMP RELATIVE cccc *see Table 2-1, "Definitions for the Condition (cccc) Bits"*																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1100				cccc				0	Offset						

PC := PC + (Offset*2) (offset is a 7-bit *signed* number from -64..+63)

JccL JUMP ABSOLUTE cccc *see Table 2-1, "Definitions for the Condition (cccc) Bits"*																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1100				cccc				1	0	Destination					

PC := Destination (destination is computed in the normal fashion for operand fields)

Rcc RET cccc *see Table 2-1, "Definitions for the Condition (cccc) Bits"*																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1100				cccc				1	0	010111					

PC := [R15]
R15++

Ccc CALL cccc *see Table 2-1, “Definitions for the Condition (cccc) Bits"*																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1010				cccc				1	0	Destination					

R15--
[R15] := Next Instruction
PC := Destination

INT																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1010				1111				0	Int vector						

R15--
[R15] := PC
PC := [int vector * 2]

This instruction allows the programmer to implement software interrupts. *Int vector* is multiplied by two, and zero extended to 16 bits.

Note: *Interrupt vectors 0 through 63 may be reserved for hardware interrupts, depending on the application.*

The condition (cccc) bits for all of the above instructions are defined in the following table.

Table 2-1. Definitions for the Condition (cccc) Bits

Condition	Mnemonic Meaning	cccc Bits	Description	Jump	CALL	RET
Z	Zero	0000	Z=1	JZ	CZ	RZ
NZ	Not Zero	0001	Z=0	JNZ	CNZ	RNZ
C / B	Carry / Borrow	0010	C=1	JC	CC	RC
NC / AE	Not Carry / Above or Equal	0011	C=0	JNC	CNC	RNC
S	Sign	0100	S=1	JS	CS	RS
NS	Not Sign	0101	S=0	JNS	CNS	RNS
O	Overflow	0110	O=1	JO	CO	RO
NO	Not Overflow	0111	O=0	JNO	CNO	RNO
A / NBE	Above / Not Below or Equal	1000	(Z=0 AND C=0)	JA	CA	RA
BE / NA	Below or Equal / Not Above	1001	(Z=1 OR C=1)	JBE	CBE	RBE
G / NLE	Greater Than / Not Less Than or Equal	1010	(O= S AND Z=0)	JG	CG	RG

Table 2-1. Definitions for the Condition (cccc) Bits (Continued)

Condition	Mnemonic Meaning	cccc Bits	Description	Jump	CALL	RET
GE / NL	Greater or Equal / Not Less Than	1011	(O=S)	JGE	CGE	RGE
L / NGE	Less Than / Not Greater or Equal	1100	(O≠S)	JL	CL	RL
LE / NG	Less Than or Equal / Not Greater Than	1101	(O≠S OR Z=1)	JLE	CLE	RLE
(not used)	---	1110				
Unconditional	Unconditionally	1111	Unconditional	JMP	CALL	RET

1.) For the JUMP mnemonics, adding an "L" to the end indicates a long or absolute jump. Adding an "S" to the end indicates a short or relative jump. If nothing is added, the assembler will choose "S" or "L."

2.) S, G, L type mnemonics use the Sign Flag and are for processing signed compares, other mnemonics like A, B etc are for unsigned compares.

2.2.3 Single Operand Operation Instructions

Since Single operand instructions do not require a source field, the format of the Single operand operation instructions is slightly different.

Instruction																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101***							[param]			Destination					

Notice that the ***opcode*** field is expanded to seven bits wide. The four most significant bits for all instructions of this class are "1101."

In addition, there is space for an optional three bit-immediate value, which is used in a manner appropriate to the instruction. The destination field functions exactly as it does in the dual operand operation instructions.

1) For the SHR, SHL, ROR, ROL, ADDI and SUBI instructions, the three-bit count or n operand is incremented by 1 before it is used. The CY16 Assembler takes this into account.

2) For the SHR, SHL, ROR, ROL, destinations can be byte-wide addresses, but shifting and rotating logic is only correct for word addresses. Byte values are treated as 16-bit values with top byte set to zeros.

3) The SHR, SHL, ROR, ROL instructions are 16-bit instructions only. 8-bit will not be supported.

SHR																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101000							Count-1			Destination					

Destination := Destination >> Count

Flags Affected: Z, C, S

Notes:

The SHR instruction shifts in sign bits.

The C flag is set with the last bit shifted out of LSB.

SHR is strictly a 16-bit operation

SHL																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101001							Count-1			Destination					

Destination := Destination << Count

Flags Affected: Z, C, S

Notes*:*

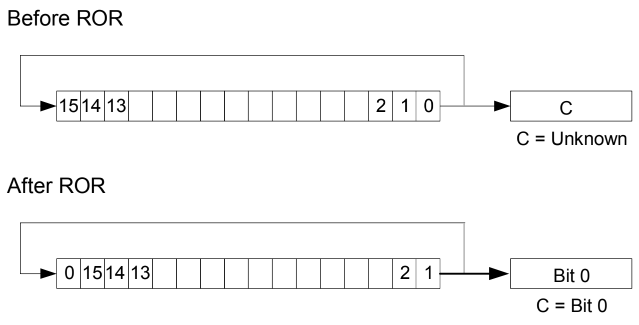
The C flag is set with the last bit shifted out of MSB.

SHL is strictly a 16-bit operation

ROR																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101010							Count-1			Destination					

Works identically to the SHR instruction, except that the LSB of *destination* is rotated into the MSB as opposed to SHR, which discards that bit.

Flags Affected: Z, C, S

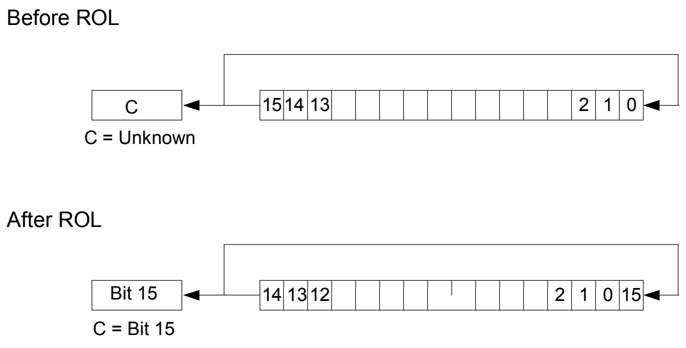


Note*: ROR is a 16-bit operation.*

ROL																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101011							Count-1			Destination					

Works identically to the SHL instruction, except that the MSB of *destination* is rotated into the LSB as opposed to SHL, which discards that bit.

Flags Affected: Z, C, S



***Note**: ROL is a 16-bit operation.*

ADDI																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101100							n-1			Destination					

Destination := Destination + n (note: n is unsigned)
Flags Affected: Z, S

SUBI																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101101							n-1			Destination					

Destination := Destination – n (note: n is unsigned)
Flags Affected: Z, S

NOT																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101111							000			Destination					

Destination := ~Destination (bitwise 1's complement negation)
Flags Affected: Z, S

NEG																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101111							001			Destination					

Destination := -Destination(2's complement negation)
Flags Affected: Z, O, C, S

CBW																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101111							100			Destination					

Sign-extends a byte in the lower eight bits of [destination] to a 16-bit signed word (integer).
Flags Affected: Z, S

Note: After excuting this instruction, the upper byte of data is destroyed.

2.2.4 Miscellaneous Instructions

STI																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101111							111			000000					

Sets Interrupt Enable Flag
Flags Affected: I

***Note**: The STI instruction takes effect 1 cycle after it is executed.*

CLI																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101111							111			000001					

Clears Interrupt Enable Flag
Flags Affected: I

STC																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101111							111			000010					

Set Carry Flag
Flags Affected: C

CLC																
Bit:	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	1101111							111			000011					

Clear Carry Flag
Flags Affected: C

2.3 Built-in Macros

For the programmer’s convenience, the CY16 Assembler implements several built-in macros. The table below shows the macros, and the mnemonics for the code that the assembler will generate for these macros.

Table 2-2. Macros and Assembler-Generated Mnemonics

Macro	Assembler will Generate
INC X	ADDI X, 1
DEC X	SUBI X, 1
PUSH X	MOV [R15], X
POP X	MOV X, [R15]

2.4 CY16 Processor Instruction Set Summary

Table 1-3. CY16 Processor Instruction Set Summary

Mnemonic	Operands	Description	Operands MSB LSB	Flags Affected	Clock Cycles	Notes
MOV	s,d	Move s to d	0000 ssss ssdd dddd	None	5	1,3
ADD	s,d	Add s to d	0001 ssss ssdd dddd	Z,C,O,S	5	1,3
ADDC	s,d	Add s to d with carry	0010 ssss ssdd dddd	Z,C,O,S	5	1,3
SUB	s,d	Subtract s from d	0011 ssss ssdd dddd	Z,C,O,S	5	1,3
SUBB	s,d	Subtract s from d with carry	0100 ssss ssdd dddd	Z,C,O,S	5	1,3
CMP	s,d	Compare d with s	0101 ssss ssdd dddd	Z,C,O,S	5	1,3
AND	s,d	AND d with s	0110 ssss ssdd dddd	Z,S	5	1,3
TEST	s,d	Bit test d with s	0111 ssss ssdd dddd	Z,S	5	1,3
OR	s,d	OR d with s	1000 ssss ssdd dddd	Z,S	5	1,3
XOR	s,d	XOR d with s	1001 ssss ssdd dddd	Z,S	5	1,3
Jcc	c,v	Jump relative on condition 'c'	1100 cccc 0ooo oooo	None	3	3
JccL	c,d	Jump absolute on condition 'c'	1100 cccc 10dd dddd	None	4	3
Rcc	c	Return on condition 'c'	1100 cccc 1001 0111	None	6	3
Ccc	c,d	Call subroutine on condition 'c'	1010 cccc 10dd dddd	None	7	3,4
Int	v	Software interrupt	1010 1111 0vvv vvvv	None	7	3,4
SHR	n,d	Shift right out of carry	1101 000n nndd dddd	Z,C,S	4	1,2,3
SHL	n,d	Shift left into carry	1101 001n nndd dddd	Z,C,S	4	1,2,3
ROR	n,d	Rotate right	1101 010n nndd dddd	Z,C,S	4	1,2,3
ROL	n,d	Rotate left	1101 011n nndd dddd	Z,C,S	4	1,2,3
ADDI	n,d	Add immediate	1101 100n nndd dddd	Z,S	4	3
SUBI	n,d	Subtract immediate	1101 101n nndd dddd	Z,S	4	3
NOT	d	1's complement	1101 1110 00dd dddd	Z,S	4	3
NEG	d	2's complement	1101 1110 01dd dddd	Z,O,C,S	4	3
CBW	d	Sign-extend d(7:0) to d(15:0)	1101 1111 00dd dddd	Z,S	4	3
STI		Enable interrupts	1101 1111 1100 0000	None	3	3
CLI		Disable interrupts	1101 1111 1100 0001	None	3	3
STC		Set carry	1101 1111 1100 0010	C	3	3
CLC		Clear carry	1101 1111 1100 0011	C	3	3

Notes:

1. The number in the "clock cycles" column reflects the number of clock cycles for register or immediate accesses. For each occurrence of other types of accesses, include the appropriate "clock adder" as listed in the Addressing Modes table below.

2. A shift of one is done in four clock cycles, each additional shift adds two more clock cycles.
3. All clock cycle values assume zero wait-states.
4. If branch is not taken, clock cycles equal 4.

Table 2-4. Opcode Field Descriptions

Field	Description
s	Source
d	Destination
c	Condition code
o	Signed offset
v	Interrupt vector
n	Count value -1

Table 2-5. Additional Instruction Clock Cycles per Addressing Mode

Addressing Mode	5	4	3	2	1	0	Clocks
Register	0	0	r	r	r	r	0
Immediate	0	1	1	1	1	1	0
Direct	1	0	b/w	1	1	1	1
Indirect	0	1	b/w	r	r	r	1
Indirect with Auto Increment	1	0	b/w	r	r	r	2
Indirect with Index	1	1	b/w	r	r	r	3
Indirect using R15	0	1	0	1	1	1	2

b/w: '1' = byte access, '0' = word access.

Indirect with auto-increment and byte-wide indirect addressing is illegal with R15.

Chapter 3 Assembly Language Reference Guide

3.1 General Information

3.1.1 Overview

The CY16 incorporated a 16-bit RSIC Processor. The processor uses two switchable banks of 16 general-purpose registers along with various memory mapped control registers. The CY16 memory space is byte addressable; however all internal operations are 16 bits.

Addressing Modes Supported

The ***source addressing*** mode:

- Register
- Immediate
- Direct
- Indirect
- Indirect with auto Increment (byte/word)
- Indirect with Index

The ***destination addressing*** mode:

- Register
- Direct
- Indirect
- Indirect with auto Increment (byte/word)
- Indirect with Index

Note: All Addressing Modes are available for both source and destination operands.

General Purpose Registers:	r0 – r15
Indirect Addressing Registers:	r8 – r15
Indirect Addressing Notation:	[r_n]
Auto Increment Notation:	[r_n++]
Preprocessor Equations:	[(nDataPtr + 5)] parentheses are not required.

*Note: In the following examples **nDataPtr** and **Table** of the GNU Assembly (see the GNUPro Auxiliary Developement Tools on file 6_auxtools.pdf for more detail) are memory variables declared in the form:*

```
nDataPtr:     .fill   1,2,0    ; 1 word, with value = 0
Table:        .fill   32,1,0   ; Table of 32 bytes, fill with value 0
Or
nDataPtr:     .fill   1,1,0    ; 1 byte, with value = 0
.align 2                       ; align in word boundary
Table:        .fill   32,1,0   ; Table of 32 bytes, fill with value 0
```

***Note:** In the following examples, registers used for each example are changed from one example to another to increase the effectiveness of the examples.*

***Notes:** The following examples are based on the GNU Assembly language in the GNUPro Auxiliary Developement Tools.*

All the calculation of the instruction cycles are based on a zero-wait state (i.e. all the code will execute in either internal RAM or internal ROM and all memory reads or memory writes are also inside the internal RAM).

Any access from internal registers (flags, interrupt enable, GPIO registers) requires one wait state (i.e. an extra cycle will be added in the calculation of the instruction cycles).

3.2 Instruction Set Description

3.2.1 DATA MOVES

MOV dest, src

Flags Set

None.

Description

Moves 8- or 16-bit data from source to destination.

Examples

MOV r0, r2 (Register)

Size:	2 Bytes
Cycles:	5

MOV r1, 0x123 (Immediate)

Size:	4 Bytes
Cycles:	5

MOV r2, [nDataPtr] (Direct)

Size:	4 Bytes
Cycles:	6

MOV [nBuffPtr], [nDataPtr] (Direct)

Size:	6 Bytes
Cycles:	7

MOV r5, (b/w)**[r10]** (InDirect)

Size:	2 Bytes
Cycles:	6

MOV r5, (b/w)**[r10++]** (InDirect w Auto Inc)

Size:	2 Bytes
Cycles:	7

MOV (b/w)**[r8],** (b/w)**[r10++]** (InDirect InDirect w Auto Inc)

Size:	2 Bytes
Cycles:	8

MOV (b/w)**[r8++],** (b/w)**[r10++]** (InDirect w Auto Inc both source and destination)

Size:	2 Bytes
Cycles:	9

MOV (b/w)**[r8++],** (b/w)**[r10]** (InDirect w Auto Inc InDirect)

Size:	2 Bytes
Cycles:	8

MOV r11, (b/w)**[r11 + Table]** (InDirect w/ Index)

Size:	4 Bytes
Cycles:	8

MOV [0xc024], 0 (w/ Index with Immediate)

Size:	6 Bytes	
Cycles:	7	(extra cycle for GPIO register access)

MOV [0x400], 0 (w/ Index with Immediate)

Size:	6 Bytes	
Cycles:	6	(0x400 is internal RAM address run at zero wait state)

Specific Code Example

```
.equ BlockAddr2, 0x1000           ; Preprocessor Constant
nCount:       .fill 1,1,0         ; Variable
.align 2
nWRP_LEN:     .short 0
nWRP_DATA:    .short 0

mov r0,b[nCount]                  ; Byte wise register and direct
mov [nWRP_LEN], 4                 ; Wordwise direct and immediate
mov [nWRP_DATA],BlockAddr2        ; Wordwise direct and immediate
mov b[(bWRP_DATA+1)],[r12++]      ; Bytewise indirect with auto increment
```

3.2.2 ADDITION

ADD dest, src ... ADDC dest, src ... ADDI dest, const

Flags Set

Z, C, O, S (Based on internal 16-bit computations only).

Description

Adds source to destination using byte or word access and sets associated result flags. ADDC also adds the carry flag for performing 32-bit addition. ADDI is immediate addition where const must be between 1 and 8 (3 bits).

Examples

ADD r7, r2 (Register)

Size: 2 Bytes
Cycles: 5

ADDI r6, 2

Size: 2 Bytes
Cycles: 4

ADD r7, 0x123 (Immediate)

Size: 4 Bytes
Cycles: 5

ADD r2, [nDataPtr] (Direct)

Size: 4 Bytes
Cycles: 6

ADD (b/w)**[r12], r0** (InDirect)

Size: 2 Bytes
Cycles: 6

ADD r1, (b/w)**[r9++]** (InDirect w/ Auto Inc)

Size: 2 Bytes
Cycles: 7

ADD r10, (b/w)**[r9 + Table]** (InDirect w/ Index)

Size: 4 Bytes
Cycles: 8

3.2.3 SUBTRACTION

SUB dest, src ... SUBB dest, src ... SUBI dest,const

Flags Set

Z, C, O, S (Based on internal 16-bit computations only).

Description

Subtracts the source from the destination using byte or word access and sets associated result flags. SUBB also subtracts the carry/borrow flag for 32-bit subtraction. SUBI is immediate subtraction where const must be between 1 and 8 (3 bits).

Examples

SUB r0, r2 (Register)

Size:	2 Bytes
Cycles:	5

SUBI r6, 2

Size:	2 Bytes
Cycles:	4

SUB r3, nDataPtr (Immediate)

Size:	4 Bytes
Cycles:	5

SUB r1, [nDataPtr] (Direct)

Size:	4 Bytes
Cycles:	6

SUB r10, (b/w)**[r8]** (InDirect)

Size:	2 Bytes
Cycles:	6

SUB (b/w)**[r9++], r5** (InDirect w/ Auto Inc)

Size:	2 Bytes
Cycles:	7

SUB r7, (b/w)**[r12 + Table]** (InDirect w/ Index)

Size:	4 Bytes
Cycles:	8

3.2.4 COMPARISON

CMP dest, src

Flags Set

Z, C, O, S (Based on internal 16-bit computations only).

Description

Compares source and destination operands. Flags = Destination - Source.

Examples

CMP r0, r2	(Register)
Size: 2 Bytes	
Cycles: 5	
CMP r1, 0x123	(Immediate)
Size: 4 Bytes	
Cycles: 5	
CMP [nDataPtr], r4	(Direct)
Size: 4 Bytes	
Cycles: 6	
CMP r1, (b/w)**[r13]**	(InDirect)
Size: 2 Bytes	
Cycles: 6	
CMP r1, (b/w)**[r9++]**	(InDirect w/ Auto Inc)
Size: 2 Bytes	
Cycles: 7	
CMP r1, (b/w)**[r9 + Table]**	(InDirect w/ Index)
Size: 4 Bytes	
Cycles: 8	

3.2.5 BIT TESTING

TEST dest, src

Flags Set

Z, S

Description

Bit-wise comparison of source and destination.

Examples

TEST r0, r2	(Register)
Size: 2 Bytes	
Cycles: 5	
TEST r1, 0x8002	(Immediate)
TEST r1, MASK	
Size: 4 Bytes	
Cycles: 5	
TEST [nDataPtr], r4	(Direct)
Size: 4 Bytes	
Cycles: 6	
TEST r1, ((b/w)**[r13]**	(InDirect)
Size: 2 Bytes	
Cycles: 6	
TEST r1, (b/w)**[r9++]**	(InDirect w/ Auto Inc)
Size: 2 Bytes	
Cycles: 7	
TEST r1, (b/w)**[r9 + Table]**	(InDirect w/ Index)
Size: 4 Bytes	
Cycles: 8	

3.2.6 LOGICAL BIT-WISE OPERATIONS

AND dest, src ... OR dest, src ... XOR dest, src

Flags Set

Z, S (Based on internal 16-bit computations only).

Description

Performs a bit-wise AND, OR or XOR operation on the source and destination operands with the result stored in destination.

Examples

All examples are the same for OR and XOR.

AND r0, r2 (Register)

Size:	2 Bytes
Cycles:	5

AND r1, 0xf80 (Immediate)

Size:	4 Bytes
Cycles:	5

AND r4, [nMask] (Direct)

Size:	4 Bytes
Cycles:	6

AND r1, (b/w)**[r9]** (InDirect)

Size:	2 Bytes
Cycles:	6

AND r1, (b/w)**[r9++]** (InDirect w/ Auto Inc)

Size:	2 Bytes
Cycles:	7

AND r1, (b/w)**[r11 + Table]** (InDirect w/ Index)

Size:	4 Bytes
Cycles:	8

3.2.7 BIT SHIFTING

SHR dest, const ... SHL dest, const

Flags Set

Z, C, S (Based on internal 16-bit computations only).

Description

Performs a bit-wise shifting (right or left). The **const shift value** must be in the range of 1 to 8. Bits which are shifted past the MSB or LSB are lost. SHR shifts in sign bits. The C flag is set when the last bit is shifted out of the LSB.

Examples

SHR r0, r2 (Invalid Instruction)

SHL r1, 1 (Immediate)

Size: 2 Bytes
Cycles: 2 + (**const**)*2 = 4

SHL r1, 2 (Immediate)

Size: 2 Bytes
Cycles: 2 + (**const**)*2 = 6

SHL r1, 8 (Immediate)

Size: 2 Bytes
Cycles: 2 + (**const**)*2 = 18

SHL (b/w)**[r12], 1** (InDirect)

Size: 2 Bytes
Cycles: 5

SHR (b/w)**[r9++], 1** (InDirect w/ Auto Inc)

Size: 2 Bytes
Cycles: 6

SHL (b/w)**[r9 + Table], 1** (InDirect w/ Index)

Size: 4 Bytes
Cycles: 7

3.2.8 BIT ROTATION

ROR dest, const ... ROL dest, const

Flags Set

Z, C, S (Based on internal 16-bit computations only).

Description

Performs a bit-wise rotation (right or left). The **const shift value** must be in the range of 1 to 8. Bits which are shifted past the MSB or LSB wrapped around, unlike the shift instructions.

Examples

ROR r0, r2 (Invalid instruction)

ROR r1, 1 (Immediate)

Size:	2 Bytes
Cycles:	2 + (**const**)*2 = 4

ROL r1, 2 (Immediate)

Size:	2 Bytes
Cycles:	2 + (**const**)*2 = 6

ROL r1, 8 (Immediate)

Size:	2 Bytes
Cycles:	2 + (**const**)*2 = 18

ROL (b/w)**[r12], 1** (InDirect)

Size:	2 Bytes
Cycles:	5

ROR (b/w)**[r9++], 1** (InDirect w/ Auto Inc)

Size:	2 Bytes
Cycles:	6

ROL (b/w)**[r9 + Table], 1** (InDirect w/ Index)

Size:	4 Bytes
Cycles:	7

3.2.9 1’s Compliment

NOT dest

Flags Set

Z, S (Based on internal 16-bit computations only).

Description

Performs a 1’s compliment on the destination data.

Examples

NOT r0 (Register)

Size:	2 Bytes
Cycles:	4

NOT (b/w)**[nDataPtr]** (InDirect)

Size:	4 Bytes
Cycles:	5

NOT (b/w)**[r9++]** (InDirect w/ Auto Inc)

Size:	2 Bytes
Cycles:	6

3.2.10 2's Compliment

NEG dest

Flags Set

Z, O,C,S (Based on internal 16-bit computations only).

Description

Performs a 12s compliment on the destination data.

Examples

NEG r0	(Register)
Size: 2 Bytes	
Cycles: 4	
NEG (b/w)**[nDataPtr]**	(InDirect)
Size: 4 Bytes	
Cycles: 5	
NEG (b/w)**[r9++]**	(InDirect w/ Auto Inc)
Size: 2 Bytes	
Cycles: 6	

3.2.11 Program Branching

JMP address

Flags Set

None.

Description

Unconditional jump to address. The assembler automatically detects short and long jumps. Addresses in range –64 to +63 are short, otherwise long is used.

Examples

JMP TestLoop

Size:	2 Bytes, 4 if long
Cycles:	3 short, 4 long

Specific Code Example

```
;--- Init L2P_Table to 0xfff ---
mov r9, L2P_Table
mov r1,[nTotalBlocks]         ; For All Blocks
lp_0:
   mov [r9++],0xffff
   dec r1
jnz lp_0
```

Conditional Versions: jz, jnz, jc, jnc, js, jns, jo, jno, ja, jbe, jg, jge, jl, jle

3.2.12 Subroutine Calling

CALL address

Flags Set

None.

Description

Calls an assembly subroutine.

Examples

CALL TestFunc

Size:	4 Bytes
Cycles:	7 (4 if conditional and branch not taken)

Conditional Versions: cz, cnz, cc, cnc, cs, cns, co, cno, ca, cbe, cg, cge, cl, cle

3.2.13 Subroutine Return

RET

Flags Set

None.

Description

Returns from subroutine.

Examples

RET

Size:	2 Bytes
Cycles:	6

Conditional Versions: rz, rnz, rc, rnc, rs, rns, ro, rno, ra, rbe, rg, rge, rl, rle

3.2.14 Software Interrupt

INT const

Flags Set

None.

Description

Triggers software interrupt with index = const.

Examples

INT 72

Size:	2 Bytes
Cycles:	7 (4 if conditional and branch not taken)

3.2.15 Set Interrupt Enable Flag

STI

Flags Set

I

Description

Enables hardware interrupts. Any interrupts pending will be serviced immediately (clock cycle 4).

Examples

STI

Size:	2 Bytes
Cycles:	3

3.2.16 Clear Interrupt Enable Flag

CLI

Flags Set

I

Description

Disables hardware interrupts. Any interrupts following will be latched.

Examples

CLI

Size:	2 Bytes
Cycles:	3

3.2.17 Set Carry Flag

STC

Flags Set

C

Description

Sets the carry flag.

Examples

STC

Size:	2 Bytes
Cycles:	3

3.2.18 Clear Carry Flag

CLC

Flags Set

C

Description

Clears the carry flag.

Examples

CLC

Size:	2 Bytes
Cycles:	3

Appendix A
Definitions

Term	Definition
R0-R15	CY16 Registers: R0-R7 Data registers or general-purpose registers R8-R14 Address/Data registers, or general-purpose registers R15 Stack pointer register
R/W	Read/Write
CY16	The CY16 is a multiport USB1.1 controller, which provides multiple functions on a single chip.
USB	**U**niversal **S**erial **B**us

Appendix B

Revision History

Name and Version	Date Issue	Comments
Rev 1.0	1/21/2003	boo
Rev 1.1	7/2/2003	sbn