Y180

8-bit Microprocessor
Synthesizable
Verilog HDL Model

User Manual

Disclaimer

Systemyde International Corporation reserves the right to make changes at any time, without notice, to improve design or performance and provide the best product possible. Systemyde International Corporation makes no warrant for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make any commitment to update the information contained herein.

Systemyde International Corporation products are not authorized for use in life support devices or systems unless a specific written agreement pertaining to such use is executed between the manufacturer and the President of Systemyde International Corporation. Nothing contained herein shall be construed as a recommendation to use any product in violation of existing patents, copyrights or other rights of third parties. No license is granted by implication or otherwise under any patent, patent rights or other rights, of Systemyde International Corporation. All trademarks are trademarks of their respective companies.

Systemyde International Corporation
http://www.systemyde.com
y180@systemyde.com

Table of Contents

Table of Contents (continued)

Table of Contents (continued)

1 Introduction

The Y180 is a synthesizable Verilog HDL model of the Z80180 CPU. It is software and hardware compatible with the Z80180 CPU and is software compatible with several other industry-standard processors. The Y180 is an original design, based on publically available documentation, that employs design techniques suitable for a technology-independent implementation. It is a fully synchronous design that does not use 3-state busses. The design is structured in a way that allows its use either with or without modification by the customer. The combinatorial logic portions of the design may be implemented in either random logic or as a PLA, and control signals are treated symbolically in the design to allow either encoded or unencoded implementations (the default is encoded). The Y180 is accompanied by full design documentation, in the form of a large spreadsheet, which describes nearly every facet of the internal operation of the processor. This provides knowledgable users the opportunity to customize the design for unique application requirements.

The Y180 is a powerful medium-performance processor that executes 181 instructions and includes an undefined opcode trap for illegal opcodes. The device contains a full complement of 8-bit arithmetic and logical instructions, and enough 16-bit instructions to properly handle the 16-bit address range. Included are bit manipulation instructions as well as an 8x8 multiply instruction. The device allows for other bus masters and includes a powerful vectored interrupt capability. The Y180 can be easily integrated with RAM, ROM or other application-specific logic to create a single-chip product. The technology-independent nature of the design provides the full spectrum of design alternatives relative to cost, power consumption and speed. Although currently limited to 8-bit data and 16-bit addresses, the architecture of the Y180 can be upgraded to 16-bit data and 32-bit addresses with relative ease.

The Y180 is written in Verilog HDL and can be synthesized using any Verilog-compatible logic synthesizer. The Y180 package includes full design documentation, including a Verilog simulation and test suite.

2 Features

* Fully functional synthesizable Verilog HDL model of the Z80180 CPU

* Vendor and technology independent

* Software compatible with several industry-standard processors

* 181 Instructions, plus an undefined opcode trap

* Eight addressing modes

* 64K byte addressing capability

* 8 bit ALU with bit, byte and BCD operations

* 8x8 multiply instruction

* Powerful vectored interrupt capability

* Static, fully synchronous design

* Designed without 3-state busses

* Easily modified external interface

* Architectural upgrade path to 16 or 32 bits possible

* Full design documentation included

* Verilog simulation and test suite included

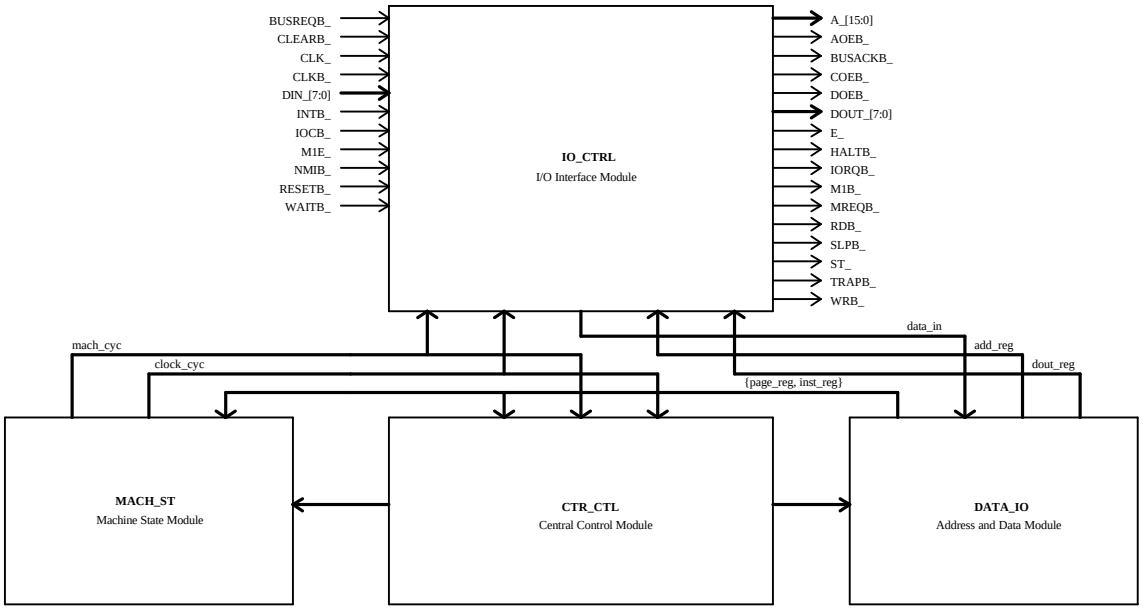
3 Functional Description

The Y180 is a general-purpose 8-bit microprocessor that is compatible with the Zilog Z80180 CPU. The device contains an 8-bit ALU, numerous 8- and 16-bit registers, a 64K byte addressing range, and a powerful vectored interrupt capability. The device executes 181 instructions, and performs an undefined opcode trap on all illegal instructions. The Y180 is completely software compatible with several industry-standard processors.

The Y180 is designed without using 3-state buses internally for maximum technology independence, and is a static, fully synchronous design. Architectural upgrades to a wider ALU, wider registers, or a wider address bus are possible. The Y180 is supplied in the form of a synthesizable Verilog HDL model, which is independent of technology, clock speed (within the limits of the chosen technology), and vendor.

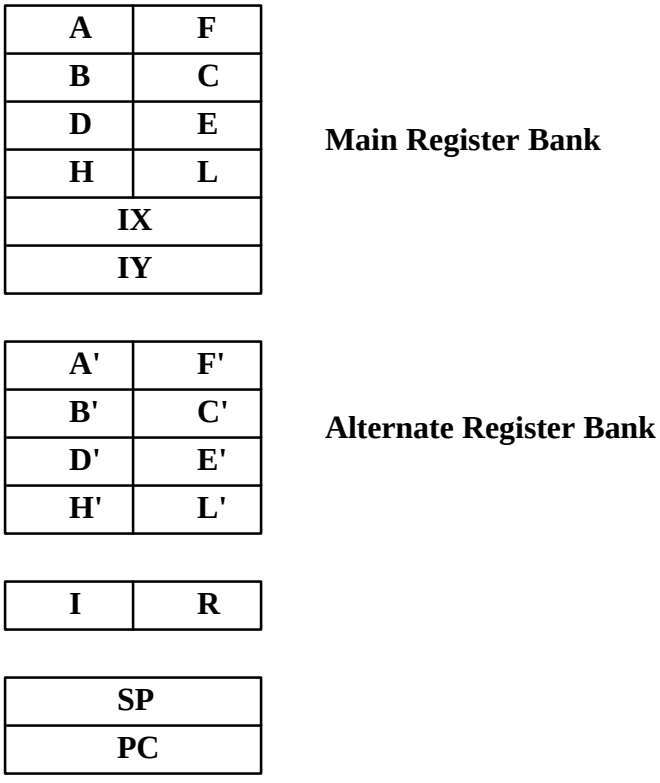
3.1 Block Diagram

The figure below shows a simplified block diagram of the Y180, organized in the same fashion as the Verilog HDL model is organized. The I/O Interface Module controls all of the pins of the Y180, and translates the internal busses and signals into the externally visible pins. The Machine State Module contains the machine cycle and clock cycle state machines, which control the sequence and timing of everything that happens within the Y180. The Central Control Module decodes the instruction and state information to generate all of the internal control signals. And the Address and Data Module contains the actual address and data manipulation portions of the Y180, including the ALU, the register file, and the various busses and special purpose registers.



3.2 Register Description

The figure below shows the registers contained in the Y180 that are visible to the programmer. The main registers have both a primary and an alternate version. The primary register set consists of A, F, B, C, D, E, H, and L, while the alternate register set consists of A', F', B', C', D', E', H', and L'. At any given time only one bank is active, and care must be used when switching between banks, as there is no way for the programmer to check which bank is active. The accumulator, A, is the destination for all 8-bit arithmetic and logic operations, while the Flag register F contains the flag results of arithmetic and logic operations. The other general-purpose registers can be paired, BC or DE or HL, to form 16-bit registers. There are two index registers, IX and IY, used for indexed addressing mode. The I register holds the upper eight bits of the interrupt vector table address for use in Interrupt Mode 2. The R register is left over from the original Z80 architecture, where it was used to hold a refresh address for DRAMs. In the Y180 it is just another general-purpose register. The Stack pointer, SP, holds the address of the stack, and the Program Counter, PC, holds the address of the currently executing instruction.



3.3 Flags Description

The figure below shows the flags contained in the F register, which report the results of instruction execution.

S	Z	U5	H	U3	P/V	N	C

S (Sign) The Sign flag stores the most significant bit of the result. This is used with signed arithmetic, where the MSB is zero for positive numbers and one for negative numbers.

Z (Zero) The Zero flag is set to one if the result of the operation is 0.

U5 (User) This is a user-defined flag. It is difficult to use however, because the only way to access it is to Push the AF register pair onto the stack and then Pop it back into some other register pair before testing the bit.

H (Half-Carry) The Half-Carry flag is used only by the DAA (Decimal Adjust Accumulator) instruction to properly adjust the result of an arithmetic operation on BCD numbers.

U3 (User) This is a user-defined flag. It is difficult to use however, because the only way to access it is to Push the AF register pair onto the stack and then Pop it back into some other register pair before testing the bit.

P/V (Parity/Overflow) The Parity/Overflow flag reports the parity of the result for logical operations, with the flag set to one if the result has even parity and zero if the result has odd parity. This bit reports the overflow status of arithmetic operations. Overflow occurs when the two operands have the same sign but the sign of the result is different. This means that the actual result cannot be represented in the eight or sixteen bits allocated for the result.

N (Negative) The Negative flag records the type of the last arithmetic operation (add or subtract) for use with the DAA instruction. The bit is set to one for subtract operations and set to zero for add operations.

C (Carry) The Carry flag is set to one whenever there is a carry or borrow from the most significant bit of the result of an arithmetic operation. This is useful for implementing multiple precision arithmetic in software.

3.4 Instruction Maps

The following sections contain the opcode maps for the Y180. The most significant nibble is indexed vertically in the tables, while the least-significant nibble is indexed horizontally in the tables. Shaded opcodes are invalid and attempted execution of these opcodes will result in a Trap. In these maps, d is an 8-bit signed displacement, e is an 8-bit signed relative address, n is an 8-bit constant, and mn is a 16-bit constant.

3.4.1 Main Code Page

This table shows the main code page for the Y180. These instructions are all one byte long unless they contain immediate data or addresses. The four bytes marked as esc (for escape) are the first byte of multi-byte instructions, which are shown in subsequent tables.

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LD BC,mn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA	EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	DJNZ e	LD DE,mn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA	JR e	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ,e	LD HL,mn	LD (mn),HL	INC HL	INC H	DEC H	LD H,n	DAA	JR Z,e	ADD HL,HL	LD HL,(mn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC,e	LD SP,mn	LD (mn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF	JR C,e	ADD HL,SP	LD A,(mn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C	RET NZ	POP BC	JP NZ,mn	JP mn	CALL NZ,mn	PUSH BC	ADD A,n	RST 0	RET Z	RET	JP Z,mn	**esc**	CALL Z,mn	CALL nn	ADC A,n	RST 1
D	RET NC	POP DE	JP NC,mn	OUT (n),A	CALL NC,mn	PUSH DE	SUB n	RST 2	RET C	EXX	JP C,mn	IN A,(n)	CALL C,mn	**esc**	SBC A,n	RST 3
E	RET PO	POP HL	JP PO,mn	EX (SP),HL	CALL PO,mn	PUSH HL	AND n	RST 4	RET PE	JP (HL)	JP PE,mn	EX DE,HL	CALL PE,mn	**esc**	XOR n	RST 5
F	RET P	POP AF	JP P,mn	DI	CALL P,mn	PUSH AF	OR n	RST 6	RET M	LD SP,HL	JP M,mn	EI	CALL M,mn	**esc**	CP n	RST 7

3.4.2 ED Code Page

This table shows the code page for instructions whose first byte is EDh. These are miscellaneous instructions that will usually not be used as often as those on the main code page.

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	IN0 B,(n)	OUT0 (n),B			TST B				IN0 C,(n)	OUT0 (n),C			TST C			
1	IN0 D,(n)	OUT0 (n),D			TST D				IN0 E,(n)	OUT0 (n),E			TST E			
2	IN0 H,(n)	OUT0 (n),H			TST H				IN0 L,(n)	OUT0 (n),L			TST L			
3					TST (HL)				IN0 A,(n)	OUT0 (n),A			TST A			
4	IN B,(C)	OUT (C),B	SBC HL,BC	LD (mn),BC	NEG	RETN	IM 0	LD I,A	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(mn)	MLT BC	RETI		LD R,A
5	IN D,(C)	OUT (C),D	SBC HL,DE	LD (mn),DE			IM 1	LD A,I	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(mn)	MLT DE		IM 2	LD A,R
6	IN H,(C)	OUT (C),H	SBC HL,HL	LD (mn),HL	TST n			RRD	IN L,(C)	OUT (C),L	ADC HL,HL	LD HL,(mn)	MLT HL			RLD
7			SBC HL,SP	LD (mn),SP	TSTIO n		SLP		IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(mn)	MLT SP			
8				OTIM								OTDM				
9				OTIMR								OTDMR				
A	LDI	CPI	INI	OUTI					LDD	CPD	IND	OUTD				
B	LDIR	CPIR	INIR	OTIR					LDDR	CPDR	INDR	OTDR				
C																
D																
E																
F																

3.4.3 DD Code Page

This table shows the code page for instructions whose first byte is DDh. All instructions on this code page imply the use of the IX register in one way or another. Note that wherever the HL register is used in a main code page instruction, the corresponding instruction on this code page uses either the IX register, or the indexed addressing mode using the IX register.

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										ADD IX,BC						
1										ADD IX,DE						
2		LD IX,mn	LD (mn),IX	INC IX						ADD IX,IX	LD IX,(mn)	DEC IX				
3					INC (IX+d)	DEC (IX+d)	LD (IX+d),n			ADD IX,SP						
4							LD B,(IX+d)								LD C,(IX+d)	
5							LD D,(IX+d)								LD E,(IX+d)	
6							LD H,(IX+d)								LD L,(IX+d)	
7	LD (IX+d),B	LD (IX+d),C	LD (IX+d),D	LD (IX+d),E	LD (IX+d),H	LD (IX+d),L		LD (IX+d),A							LD A,(IX+d)	
8							ADD A,(IX+d)								ADC A,(IX+d)	
9							SUB (IX+d)								SBC A,(IX+d)	
A							AND (IX+d)								XOR (IX+d)	
B							OR (IX+d)								CP (IX+d)	
C												**esc**				
D																
E		POP IX		EX (SP),IX		PUSH IX				JP (IX)						
F										LD SP,IX						

3.4.4 FD Code Page

This table shows the code page for instructions whose first byte is FDh. All instructions on this code page imply the use of the IY register in one way or another. This code page is identical to the DD code page with the IY register substituted for the IX register.

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										ADD IY,BC						
1										ADD IY,DE						
2		LD IY,mn	LD (mn),IY	INC IY						ADD IY,IY	LD IY,(mn)	DEC IY				
3					INC (IY+d)	DEC (IY+d)	LD (IY+d),n			ADD IX,SP						
4							LD B,(IY+d)								LD C,(IY+d)	
5							LD D,(IY+d)								LD E,(IY+d)	
6							LD H,(IY+d)								LD L,(IY+d)	
7	LD (IY+d),B	LD (IY+d),C	LD (IY+d),D	LD (IY+d),E	LD (IY+d),H	LD (IY+d),L		LD (IY+d),A							LD A,(IY+d)	
8							ADD A,(IY+d)								ADC A,(IY+d)	
9							SUB (IY+d)								SBC A,(IY+d)	
A							AND (IY+d)								XOR (IY+d)	
B							OR (IY+d)								CP (IY+d)	
C												**esc**				
D																
E		POP IY		EX (SP),IY		PUSH IY				JP (IY)						
F										LD SP,IY						

3.4.5 CB Code Page

This table shows the code page for instructions whose first byte is CBh. The instructions on this code page are the majority of the shift, rotate and bit manipulation instructions.

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3									SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
4	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

3.4.6 DD-CB Code Page

This table shows the code page for instructions whose first two bytes are DDh, followed by CBh. All instructions on this code page imply the use of the IX register in one way or another. Note that wherever the HL register is used in a CB code page instruction, the corresponding instruction on this code page uses either the IX register, or the indexed addressing mode using the IX register.

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0							RLC (IX+d)								RRC (IX+d)	
1							RL (IX+d)								RR (IX+d)	
2							SLA (IX+d)								SRA (IX+d)	
3															SRL (IX+d)	
4							BIT 0,(IX+d)								BIT 1,(IX+d)	
5							BIT 2,(IX+d)								BIT 3,(IX+d)	
6							BIT 4,(IX+d)								BIT 5,(IX+d)	
7							BIT 6,(IX+d)								BIT 7,(IX+d)	
8							RES 0,(IX+d)								RES 1,(IX+d)	
9							RES 2,(IX+d)								RES 3,(IX+d)	
A							RES 4,(IX+d)								RES 5,(IX+d)	
B							RES 6,(IX+d)								RES 7,(IX+d)	
C							SET 0,(IX+d)								SET 1,(IX+d)	
D							SET 2,(IX+d)								SET 3,(IX+d)	
E							SET 4,(IX+d)								SET 5,(IX+d)	
F							SET 6,(IX+d)								SET 7,(IX+d)	

3.4.7 FD-CB Code Page

This table shows the code page for instructions whose first two bytes are FDh, followed by CBh. All instructions on this code page imply the use of the IY register in one way or another. This code page is identical to the DD-CB code page with the IY register substituted for the IX register.

\LSB MSB\	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0							RLC (IY+d)								RRC (IY+d)	
1							RL (IY+d)								RR (IY+d)	
2							SLA (IY+d)								SRA (IY+d)	
3															SRL (IY+d)	
4							BIT 0,(IY+d)								BIT 1,(IY+d)	
5							BIT 2,(IY+d)								BIT 3,(IY+d)	
6							BIT 4,(IY+d)								BIT 5,(IY+d)	
7							BIT 6,(IY+d)								BIT 7,(IY+d)	
8							RES 0,(IY+d)								RES 1,(IY+d)	
9							RES 2,(IY+d)								RES 3,(IY+d)	
A							RES 4,(IY+d)								RES 5,(IY+d)	
B							RES 6,(IY+d)								RES 7,(IY+d)	
C							SET 0,(IY+d)								SET 1,(IY+d)	
D							SET 2,(IY+d)								SET 3,(IY+d)	
E							SET 4,(IY+d)								SET 5,(IY+d)	
F							SET 6,(IY+d)								SET 7,(IY+d)	

3.5 Execution Tables

The tables below show the operation of the Y180 in detail for all instructions and exception conditions. These tables are a part of the spreadsheet included in the full electronic documentation for the Y180, which contains things like ALU operations, bus contents, internal register addresses, etc. The tables below should be sufficient for the majority of users of the Y180, but if you intend to modify the Y180 for your application, or merely want to understand the internal workings of the design, refer to the full spreadsheet for more detailed information.

3.5.1 Execution Table Conventions

The conventions used in the instruction, opcode and operation columns of the execution tables are as follows:

b	bit select (000 = bit 0, 001 = bit 1, 010 = bit 2, 011 = bit 3, 100 = bit 4, 101 = bit 5, 110 = bit 6, 111 = bit 7)
cc	condition code select (00 = NZ, 01 = Z, 10 = NC, 11 = C)
d	8-bit (signed) displacement
dd	word register select (00 = BC, 01 = DE, 10 = IX, 11 = SP)
e	8-bit (signed) displacement
f	condition code select (000 = NZ, 001 = Z, 010 = NC, 011 = C, 100 = PO, 101 = PE, 110 = P, 111 = M)
m	MSB of a 16-bit constant
mn	16-bit constant
n	8-bit constant or LSB of a 16-bit constant
r, r'	byte register select (000 = B, 001 = C, 010 = D, 011 = E, 100 = H, 101 = L, 111 = A)
ss	word register select (00 = BC, 01 = DE, 10 = HL, 11 = SP)
v	Restart address select (000 = 0000h, 001 = 0008h, 010 = 0010h, 011 = 0018h, 100 = 0020h, 101 = 0028h, 110 = 0030h, 111 = 0038h)
xx	word register select (00 = BC, 01 = DE, 10 = IX, 11 = SP)
yy	word register select (00 = BC, 01 = DE, 10 = IY, 11 = SP)
zz	word register select (00 = BC, 01 = DE, 10 = HL, 11 = AF)

The conventions used in the flag columns of the execution tables are as follows:

-	No change
*	Updated per convention
0, 1	Reset to zero or set to one
IE	Set to value of IEF1 bit
P, V	Reports the parity (P) or overflow (V) status of the result

3.5.2 Instruction Opcode, Timing and Operation

The execution table below shows the instruction or exception, the opcode, the addressing mode, the number of machine cycles, the number and organization of the clock cycles, the flags affected by the instruction, and the operation performed by the instruction or exception.

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Addr Mode	Mach State	Clock cycles	S	Z	H	P V	N	C	Operation
ADC A,(HL)	10001110				reg ind	2	6 (3,3)	*	*	*	V	0	*	A = A + (HL) + CF
ADC A,(IX+d)	11011101	10001110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	0	*	A = A + (IX+d) + CF
ADC A,(IY+d)	11111101	10001110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	0	*	A = A + (IY+d) + CF
ADC A,n	11001110	----n---			immed	2	6 (3,3)	*	*	*	V	0	*	A = A + n + CF
ADC A,r	10001-r-				reg	2	4 (3,1)	*	*	*	V	0	*	A = A + r + CF
ADC HL,ss	11101101	01ss1010			reg	3	10 (3,3,4)	*	*	*	V	0	*	HL = HL + ss + CF
ADD A,(HL)	10000110				reg ind	2	6 (3,3)	*	*	*	V	0	*	A = A + (HL)
ADD A,(IX+d)	11011101	10000110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	0	*	A = A + (IX+d)
ADD A,(IY+d)	11111101	10000110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	0	*	A = A + (IY+d)
ADD A,n	11000110	----n---			immed	2	6 (3,3)	*	*	*	V	0	*	A = A + n
ADD A,r	10000-r-				reg	2	4 (3,1)	*	*	*	V	0	*	A = A + r
ADD HL,ss	00ss1001				reg	2	7 (3,4)	*	*	*	V	0	*	HL = HL + ss
ADD IX,xx	11011101	00xx1001			reg	3	10 (3,3,4)	*	*	*	V	0	*	IX = IX + xx
ADD IY,yy	11111101	00yy1001			reg	3	10 (3,3,4)	*	*	*	V	0	*	IY = IY + yy
AND (HL)	10100110				reg ind	2	6 (3,3)	*	*	1	P	0	0	A = A & (HL)
AND (IX+d)	11011101	10100110	----d---		index	5	14 (3,3,3,2,3)	*	*	1	P	0	0	A = A & (IX+d)
AND (IY+d)	11111101	10100110	----d---		index	5	14 (3,3,3,2,3)	*	*	1	P	0	0	A = A & (IY+d)
AND n	11100110	----n---			immed	2	6 (3,3)	*	*	1	P	0	0	A = A & n
AND r	10100-r-				reg	2	4 (3,1)	*	*	1	P	0	0	A = A & r
BIT b,(HL)	11001011	01-b-110			reg ind	3	9 (3,3,3)	-	*	1	-	0	-	(HL) & bit
BIT b,(IX+d))	11011101	11001011	----d---	01-b-110	index	5	15 (3,3,3,3,3)	-	*	1	-	0	-	(IX+d) & bit
BIT b,(IY+d))	11111101	11001011	----d---	01-b-110	index	5	15 (3,3,3,3,3)	-	*	1	-	0	-	(IY+d) & bit
BIT b,r	11001011	01-b--r-			reg	2	6 (3,3)	-	*	1	-	0	-	r & bit
CALL f,mn	11-f-100	----n---	----m---		immed	2 (F) 6 (T)	6 (3,3) 16 (3,3,3,1,3,3)	-	-	-	-	-	-	if {f} (SP-1) = PCH; (SP-2) = PCL; PC = mn; SP = SP-2
CALL mn	11001101	----n---	----m---		immed	6	16 (3,3,3,1,3,3)	-	-	-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; PC = mn; SP = SP-2
CCF	00111111				none	1	3	-	-	0	-	0	*	CF = ~CF
CP (HL)	10111110				reg ind	2	6 (3,3)	*	*	*	V	1	*	A - (HL)
CP (IX+d)	11011101	10111110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	1	*	A - (IX+d)
CP (IY+d)	11111101	10111110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	1	*	A - (IY+d)
CP n	11111110	----n---			immed	2	6 (3,3)	*	*	*	V	1	*	A - n
CP r	10111-r-				reg	2	4 (3,1)	*	*	*	V	1	*	A - r
CPD	11101101	10101001			implied	4	12 (3,3,3,3)	*	*	*	*	1	-	A - (HL); BC = BC-1; HL = HL-1
CPDR	11101101	10111001			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	*	*	*	*	1	-	if {(BC != 0) | (A != (HL))} repeat: A - (HL); BC = BC-1; HL = HL-1
CPI	11101101	10100001			implied	4	12 (3,3,3,3)	*	*	*	*	1	-	A - (HL); BC = BC-1; HL = HL+1
CPIR	11101101	10110001			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	*	*	*	*	1	-	if {(BC != 0) | (A != (HL))} repeat: A - (HL); BC = BC-1; HL = HL+1
CPL	00101111				implied	1	3	-	-	1	-	1	-	A = ~A
DAA	00100111				implied	2	4 (3,1)	*	*	*	P	-	*	Decimal Adjust Accumulator
DEC (HL)	00110101				reg ind	4	10 (3,3,1,3)	*	*	*	V	1	*	(HL) = (HL) - 1
DEC (IX+d)	11011101	00110101	----d---		index	7	18 (3,3,3,2,3,1,3)	*	*	*	V	1	*	(IX+d) = (IX+d) -1
DEC (IY+d)	11111101	00110101	----d---		index	7	18 (3,3,3,2,3,1,3)	*	*	*	V	1	*	(IY+d) = (IY+d) -1

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Addr Mode	Mach State	Clock cycles	S	Z	H	P V	N	C	Operation
DEC IX	11011101	00101011			reg	3	7 (3,3,1)	-	-	-	-	-	-	IX = IX - 1
DEC IY	11111101	00101011			reg	3	7 (3,3,1)	-	-	-	-	-	-	IY = IY - 1
DEC r	00-r-101				reg	2	4 (3,1)	*	*	*	V	1	*	r = r - 1
DEC ss	00ss1011				reg	2	4 (3,1)	-	-	-	-	-	-	ss = ss - 1
DI	11110011				none	1	3	-	-	-	-	-	-	IEF1 = 0; IEF2 = 0
DJNZ e	00010000	--(e-2)-			relative	3 (F) 4 (T)	7 (3,1,3) 9 (3,1,3,2)	-	-	-	-	-	-	B = B-1; if {B != 0} PC = PC + e
EI	11111011				none	1	3	-	-	-	-	-	-	IEF1 =1; IEF2 = 1
EX (SP),HL	11100011				implied	6	16 (3,3,3,1,3,3)	-	-	-	-	-	-	H <-> (SP+1); L <-> (SP)
EX (SP),IX	11011101	11100011			implied	7	19 (3,3,3,3,1,3,3)	-	-	-	-	-	-	IXH <-> (SP+1); IXL <-> (SP)
EX (SP),IY	11111101	11100011			implied	7	19 (3,3,3,3,1,3,3)	-	-	-	-	-	-	IYH <-> (SP+1); IYL <-> (SP)
EX AF,AF'	00001000				implied	2	4 (3,1)	-	-	-	-	-	-	AF <-> AF'
EX DE,HL	11101011				implied	1	3	-	-	-	-	-	-	DE <-> HL
EXX	11011001				implied	1	3	-	-	-	-	-	-	BC <-> BC'; DE <-> DE'; HL <-> HL'
HALT	01110110				none	1	3	-	-	-	-	-	-	CPU halted
IM 0	11101101	01000110			none	2	6 (3,3)	-	-	-	-	-	-	Interrupt mode 0
IM 1	11101101	01010110			none	2	6 (3,3)	-	-	-	-	-	-	Interrupt mode 1
IM 2	11101101	01011110			none	2	6 (3,3)	-	-	-	-	-	-	Interrupt mode 2
IN A,(n)	11011011	----n---			direct	3	9 (3,3,3)	-	-	-	-	-	-	A = (An)
IN r,(C)	11101101	01-r-000			indirect	3	9 (3,3,3)	*	*	0	P	0	-	r = (BC)
IN0 r,(n)	11101101	00-r-000	----n---		direct	4	12 (3,3,3,3)	*	*	0	P	0	-	r = (n)
INC (HL)	00110100				reg ind	4	10 (3,3,1,3)	*	*	*	V	0	*	(HL) = (HL) + 1
INC (IX+d)	11011101	00110100	----d---		index	7	18 (3,3,3,2,3,1,3)	*	*	*	V	0	*	(IX+d) = (IX+d) + 1
INC (IY+d)	11111101	00110100	----d---		index	7	18 (3,3,3,2,3,1,3)	*	*	*	V	0	*	(IY+d) = (IY+d) + 1
INC IX	11011101	00100011			reg	3	7 (3,3,1)	-	-	-	-	-	-	IX = IX + 1
INC IY	11111101	00100011			reg	3	7 (3,3,1)	-	-	-	-	-	-	IY = IY + 1
INC r	00-r-100				reg	2	4 (3,1)	*	*	*	V	0	*	r = r + 1
INC ss	00ss0011				reg	2	4 (3,1)	-	-	-	-	-	-	ss = ss + 1
IND	11101101	10101010			implied	4	12 (3,3,3,3)	-	*	-	-	*	-	(HL) = (BC); HL = HL-1; B = B-1
INDR	11101101	10111010			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	-	1	-	-	*	-	if {B != 0} repeat: (HL) = (BC); HL = HL-1; B = B-1
INI	11101101	10100010			implied	4	12 (3,3,3,3)	-	*	-	-	*	-	(HL) = (BC); HL = HL+1; B = B-1
INIR	11101101	10110010			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	-	1	-	-	*	-	if {B != 0} repeat: (HL) = (BC); HL = HL+1; B = B-1
JP (HL)	11101001				implied	1	3	-	-	-	-	-	-	PC = HL
JP (IX)	11011101	11101001			implied	2	6 (3,3)	-	-	-	-	-	-	PC = IX
JP (IY)	11111101	11101001			implied	2	6 (3,3)	-	-	-	-	-	-	PC = IY
JP f,mn	11-f-010	----n---	----m---		immed	2 (F) 3 (T)	6 (3,3) 9 (3,3,3)	-	-	-	-	-	-	if {f} PC = mn
JP mn	11000011	----n---	----m---		immed	3	9 (3,3,3)	-	-	-	-	-	-	PC = mn
JR cc,j	001cc000	--(j-2)-			relative	2 (F) 3 (T)	6 (3,3) 8 (3,3,2)	-	-	-	-	-	-	if {cc} PC = PC + j
JR j	00011000	--(j-2)-			relative	3	8 (3,3,2)	-	-	-	-	-	-	PC = PC + j
LD (BC),A	00000010				implied	3	7 (3,1,3)	-	-	-	-	-	-	(BC) = A
LD (DE),A	00010010				implied	3	7 (3,1,3)	-	-	-	-	-	-	(DE) = A
LD (HL),n	00110110	----n---			immed	3	9 (3,3,3)	-	-	-	-	-	-	(HL) = n
LD (HL),r	01110-r-				reg	3	7 (3,1,3)	-	-	-	-	-	-	(HL) = r
LD (IX+d),n	11011101	00110110	----d---	----n---	immed	5	15 (3,3,3,3,3)	-	-	-	-	-	-	(IX+d) = n
LD (IX+d),r	11011101	01110-r-	----d---		reg	5	15 (3,3,3,3,3)	-	-	-	-	-	-	(IX+d) = r
LD (IY+d),n	11111101	00110110	----d---	----n---	immed	5	15 (3,3,3,3,3)	-	-	-	-	-	-	(IY+d) = n
LD (IY+d),r	11111101	01110-r-	----d---		reg	5	15 (3,3,3,3,3)	-	-	-	-	-	-	(Iy+d) = r
LD (mn),A	00110010	----n---	----m---		direct	5	13 (3,3,3,1,3)	-	-	-	-	-	-	(mn) = A

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Addr Mode	Mach State	Clock cycles	S	Z	H	P V	N	C	Operation
LD (mn),HL	00100010	----n---	----m---		direct	6	16 (3,3,3,1,3,3)	-	-	-	-	-	-	(mn) = L; (mn+1) = H
LD (mn),IX	11011101	00100010	----n---	----m---	direct	7	19 (3,3,3,3,1,3,3)	-	-	-	-	-	-	(mn) = IXL; (mn+1) = IXH
LD (mn),IY	11111101	00100010	----n---	----m---	direct	7	19 (3,3,3,3,1,3,3)	-	-	-	-	-	-	(mn) = IYL; (mn+1) = IYH
LD (mn),ss	11101101	01ss0011	----n---	----m---	direct	7	19 (3,3,3,3,1,3,3)	-	-	-	-	-	-	(mn) = ssl; (mn+1) = ssh
LD A,(BC)	00001010				implied	2	6 (3,3)	-	-	-	-	-	-	A = (BC)
LD A,(DE)	00011010				implied	2	6 (3,3)	-	-	-	-	-	-	A = (DE)
LD A,(mn)	00111010	----n---	----m---		direct	4	12 (3,3,3,3)	-	-	-	-	-	-	A = (mn)
LD A,I	11101101	01010111			implied	2	6 (3,3)	*	*	0	IE	0	-	A = I
LD A,R	11101101	01011111			implied	2	6 (3,3)	*	*	0	IE	0	-	A = R
LD dd,(mn)	11101101	01dd1011	----n---	----m---	direct	6	18 (3,3,3,3,3,3)	-	-	-	-	-	-	ddl = (mn); ddh = (mn+1)
LD dd,mn	00dd0001	----n---	----m---		direct	3	9 (3,3,3)	-	-	-	-	-	-	dd = mn
LD HL,(mn)	00101010	----n---	----m---		direct	5	15 (3,3,3,3,3)	-	-	-	-	-	-	L = (mn); H = (mn+1)
LD I,A	11101101	01000111			implied	2	6 (3,3)	-	-	-	-	-	-	I = A
LD IX,(mn)	11011101	00101010	----n---	----m---	direct	6	18 (3,3,3,3,3,3)	-	-	-	-	-	-	IXL = (mn); IXH = (mn+1)
LD IX,mn	11011101	00100001	----n---	----m---	direct	4	12 (3,3,3,3)	-	-	-	-	-	-	IX = mn
LD IY,(mn)	11111101	00101010	----n---	----m---	direct	6	18 (3,3,3,3,3,3)	-	-	-	-	-	-	IYL = (mn); IYH = (mn+1)
LD IY,mn	11111101	00100001	----n---	----m---	direct	4	12 (3,3,3,3)	-	-	-	-	-	-	IY = mn
LD r,(HL)	01-r-110				reg ind	2	6 (3,3)	-	-	-	-	-	-	r = (HL)
LD r,(IX+d)	11011101	01-r-110	----d---		index	5	14 (3,3,3,2,3)	-	-	-	-	-	-	r = (IX+d)
LD r,(IY+d)	11111101	01-r-110	----d---		index	5	14 (3,3,3,2,3)	-	-	-	-	-	-	r = (IY+d)
LD R,A	11101101	01001111			implied	2	6 (3,3)	-	-	-	-	-	-	R = A
LD r,n	00-r-110	----n---			immed	2	6 (3,3)	-	-	-	-	-	-	r = n
LD r,r'	01-r--r'				reg	2	4 (3,1)	-	-	-	-	-	-	r = r'
LD SP,HL	11111001				implied	2	4 (3,1)	-	-	-	-	-	-	SP = HL
LD SP,IX	11011101	11111001			implied	3	7 (3,3,1)	-	-	-	-	-	-	SP = IX
LD SP,IY	11111101	11111001			implied	3	7 (3,3,1)	-	-	-	-	-	-	SP = IY
LDD	11101101	10101000			implied	4	12 (3,3,3,3)	-	-	0	*	0	-	(DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1
LDDR	11101101	10111000			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	-	-	0	*	0	-	if {BC != 0} repeat: (DE) = (HL); BC = BC-1; DE = DE-1; HL = HL-1
LDI	11101101	10100000			implied	4	12 (3,3,3,3)	-	-	0	*	0	-	(DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1
LDIR	11101101	10110000			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	-	-	0	*	0	-	if {BC != 0} repeat: (DE) = (HL); BC = BC-1; DE = DE+1; HL = HL+1
MLT ww	11101101	01ww1100			reg	3	16 (3,3,10)	-	-	-	-	-	-	ww = wwl * wwh
NEG	11101101	01000100			implied	2	6 (3,3)	*	*	*	V	1	*	A = 0 - A
NOP	00000000				none	1	3	-	-	-	-	-	-	No operation
OR (HL)	10110110				reg ind	2	6 (3,3)	*	*	0	P	0	0	A = A | (HL)
OR (IX+d)	11011101	10110110	----d---		index	5	14 (3,3,3,2,3)	*	*	0	P	0	0	A = A | (IX+d)
OR (IY+d)	11111101	10110110	----d---		index	5	14 (3,3,3,2,3)	*	*	0	P	0	0	A = A | (IY+d)
OR n	11110110	----n---			immed	2	6 (3,3)	*	*	0	P	0	0	A = A | n
OR r	10110-r-				reg	2	4 (3,1)	*	*	0	P	0	0	A = A | r
OTDM	11101101	10001011			implied	6	14 (3,3,1,3,3,1)	*	*	*	P	*	*	(C) = (HL); HL = HL-1; B = B-1; C = C-1
OTDMR	11101101	10011011			implied	6 (F) 6 (T)	14 (3,3,1,3,3,1) 16 (3,3,1,3,3,3)	0	1	0	P	*	0	if {B != 0} repeat: (BC) = (HL); HL = HL-1; B = B-1; C = C-1
OTDR	11101101	10111011			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	-	1	-	-	*	-	if {B != 0} repeat: (BC) = (HL); HL = HL-1; B = B-1
OTIM	11101101	10000011			implied	6	14 (3,3,1,3,3,1)	*	*	*	P	*	*	(C) = (HL); HL = HL+1; B = B-1; C = C-1
OTIMR	11101101	10010011			implied	6 (F) 6 (T)	14 (3,3,1,3,3,1) 16 (3,3,1,3,3,3)	0	1	0	P	*	0	if {B != 0} repeat: (BC) = (HL); HL = HL+1; B = B-1; C = C-1
OTIR	11101101	10110011			implied	4 (F) 5 (T)	12 (3,3,3,3) 14 (3,3,3,3,2)	-	1	-	-	*	-	if {B != 0} repeat: (BC) = (HL); HL = HL+1; B = B-1
OUT (C),r	11101101	01-r-001			indirect	4	10 (3,3,1,3)	-	-	-	-	-	-	(BC) = r
OUT (n),A	11010011	----n---			direct	4	10 (3,3,1,3)	-	-	-	-	-	-	(An) = A
OUT0 (n),r	11101101	00-r-001	----n---		direct	5	13 (3,3,3,1,3)	-	-	-	-	-	-	(n) = r

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Addr Mode	Mach State	Clock cycles	S	Z	H	P V	N	C	Operation
OUTD	11101101	10101011			implied	4	12 (3,3,3,3)	-	*	-	-	*	-	(BC) = (HL); HL = HL-1; B = B-1
OUTI	11101101	10100011			implied	4	12 (3,3,3,3)	-	*	-	-	*	-	(BC) = (HL); HL = HL+1; B = B-1
POP IX	11011101	11100001			reg	4	12 (3,3,3,3)	-	-	-	-	-	-	IXL = (SP); IXH = (SP+1); SP = SP + 2
POP IY	11111101	11100001			reg	4	12 (3,3,3,3)	-	-	-	-	-	-	IYL = (SP); IYH = (SP+1); SP = SP + 2
POP zz	11zz0001				reg	3	9 (3,3,3)	-	-	-	-	-	-	zzl = (SP); zzh = (SP+1); SP = SP + 2
PUSH IX	11011101	11100101			reg	5	14 (3,3,2,3,3)	-	-	-	-	-	-	(SP-1) = IXH; (SP-2) = IXL; SP = SP - 2
PUSH IY	11111101	11100101			reg	5	14 (3,3,2,3,3)	-	-	-	-	-	-	(SP-1) = IYH; (SP-2) = IYL; SP = SP - 2
PUSH zz	11zz0101				reg	4	11 (3,2,3,3)	-	-	-	-	-	-	(SP-1) = zzh; (SP-2) = zzl; SP = SP - 2
RES b,(HL)	11001011	10-b-110			reg ind	5	13 (3,3,3,1,3)	-	-	-	-	-	-	(HL) = (HL) & ~bit
RES b,(IX+d)	11011101	11001011	----d---	10-b-110	index	7	19 (3,3,3,3,3,1,3)	-	-	-	-	-	-	(IX+d) = (IX+d) & ~bit
RES b,(IY+d)	11111101	11001011	----d---	10-b-110	index	7	19 (3,3,3,3,3,1,3)	-	-	-	-	-	-	(IY+d) = (IY+d) & ~bit
RES b,r	11001011	10-b--r-			reg	3	7 (3,3,1)	-	-	-	-	-	-	r = r & ~bit
RET	11001001				implied	3	9 (3,3,3)	-	-	-	-	-	-	PCL = (SP); PCH = (SP+1); SP = SP+2
RET f	11-f-000				implied	2 (F) 4 (T)	5 (3,2) 10 (3,1,3,3)	-	-	-	-	-	-	if {f} PCL = (SP); PCH = (SP+1); SP = SP+2
RETI	11101101	01001101			implied	4 (0,1) 8 (2)	12 (3,3,3,3) 22 (3,3,3,3,1,3,3,3)	-	-	-	-	-	-	PCL = (SP); PCH = (SP+1); SP = SP+2
RETN	11101101	01000101			implied	4	12 (3,3,3,3)	-	-	-	-	-	-	PCL = (SP); PCH = (SP+1); SP = SP+2; IEF2 = IEF1
RL (HL)	11001011	00010110			reg ind	5	13 (3,3,3,1,3)	*	*	0	P	0	*	{CY,(HL)} = {(HL),CY}
RL (IX+d)	11011101	11001011	----d---	00010110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	{CY,(IX+d)} = {(IX+d),CY}
RL (IY+d)	11111101	11001011	----d---	00010110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	{CY,(IY+d)} = {(IY+d),CY}
RL r	11001011	00010-r-			reg	3	7 (3,3,1)	*	*	0	P	0	*	{CY,r} = {r,CY}
RLA	00010111				implied	1	3	-	-	0	-	0	*	{CY,A} = {A,CY}
RLC (HL)	11001011	00000110			reg ind	5	13 (3,3,3,1,3)	*	*	0	P	0	*	(HL) = {(HL)[6,0],(HL)[7]}; CY = (HL)[7]
RLC (IX+d)	11011101	11001011	----d---	00000110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IX+d) = {(IX+d)[6,0],(IX+d)[7]}; CY = (IX+d)[7]
RLC (IY+d)	11111101	11001011	----d---	00000110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IY+d) = {(IY+d)[6,0],(IY+d)[7]}; CY = (IY+d)[7]
RLC r	11001011	00000-r-			reg	3	7 (3,3,1)	*	*	0	P	0	*	r = {r[6,0],r[7]}; CY = r[7]
RLCA	00000111				implied	1	3	-	-	0	-	0	*	A = {A[6,0],A[7]}; CY = A[7]
RLD	11101101	01101111			implied	5	16 (3,3,3,4,3)	*	*	0	P	0	-	A[3,0] = (HL)[7,4]; (HL) = {(HL)[3,0],A[3,0]}
RR (HL)	11001011	00011110			reg ind	5	13 (3,3,3,1,3)	*	*	0	P	0	*	{(HL),CY} = {CY,(HL)}
RR (IX+d)	11011101	11001011	----d---	00011110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	{(IX+d),CY} = {CY,(IX+d)}
RR (IY+d)	11111101	11001011	----d---	00011110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	{(IY+d),CY} = {CY,(IY+d)}
RR r	11001011	00011-r-			reg	3	7 (3,3,1)	*	*	0	P	0	*	{r,CY} = {CY,r}
RRA	00011111				implied	1	3	-	-	0	-	0	*	{A,CY} = {CY,A}
RRC (HL)	11001011	00001110			reg ind	5	13 (3,3,3,1,3)	*	*	0	P	0	*	(HL) = {(HL)[0],(HL)[7,1]}; CY = (HL)[0]
RRC (IX+d)	11011101	11001011	----d---	00001110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IX+d) = {(IX+d)[0],(IX+d)[7,1]}; CY = (IX+d)[0]
RRC (IY+d)	11111101	11001011	----d---	00001110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IY+d) = {(IY+d)[0],(IY+d)[7,1]}; CY = (IY+d)[0]
RRC r	11001011	00001-r-			reg	3	7 (3,3,1)	*	*	0	P	0	*	r = {r[0],r[7,1]}; CY = r[0]
RRCA	00001111				implied	1	3	-	-	0	-	0	*	A = {A[0],A[7,1]}; CY = A[0]
RRD	11101101	01100111			implied	5	16 (3,3,3,4,3)	*	*	0	P	0	-	A[3,0] = (HL)[3,0]; (HL) = {A[3,0],(HL)[7,4]}
RST v	11-v-111				implied	4	11 (3,2,3,3)	-	-	-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP - 2; PC = v
SBC (IX+d)	11011101	10011110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	1	*	A = A - (IX+d) - CY
SBC (IY+d)	11111101	10011110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	1	*	A = A - (IY+d) - CY
SBC A,(HL)	10011110				reg ind	2	6 (3,3)	*	*	*	V	1	*	A = A - (HL) - CY
SBC A,n	11011110	----n---			immed	2	6 (3,3)	*	*	*	V	1	*	A = A - n - CY
SBC A,r	10011-r-				reg	2	4 (3,1)	*	*	*	V	1	*	A = A - r - CY
SBC HL,ss	11101101	01ss0010			reg	3	10 (3,3,4)	*	*	*	V	1	*	HL = HL - ss - CF
SCF	00110111				none	1	3	-	-	0	-	0	1	CF = 1
SET b,(HL)	11001011	11-b-110			reg ind	4	13 (3,3,3,1,3)	-	-	-	-	-	-	(HL) = (HL) | bit
SET b,(IX+d)	11011101	11001011	----d---	11-b-110	index	7	19 (3,3,3,3,3,1,3)	-	-	-	-	-	-	(IX+d) = (IX+d) | bit
SET b,(IY+d)	11111101	11001011	----d---	11-b-110	index	7	19 (3,3,3,3,3,1,3)	-	-	-	-	-	-	(IY+d) = (IY+d) | bit
SET b,r	11001011	11-b--r-			reg	3	7 (3,3,1)	-	-	-	-	-	-	r = r | bit

Instruction	Opcode byte 1	Opcode byte 2	Opcode byte 3	Opcode byte 4	Addr Mode	Mach State	Clock cycles	S	Z	H	P V	N	C	Operation
SLA (HL)	11001011	00100110			reg ind	5	13 (3,3,3,1,3)	*	*	0	P	0	*	(HL) = {(HL)[6,0],0}; CY = (HL)[7]
SLA (IX+d)	11011101	11001011	----d---	00100110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IX+d) = {(IX+d)[6,0],0}; CY = (IX+d)[7]
SLA (IY+d)	11111101	11001011	----d---	00100110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IY+d) = {(IY+d)[6,0],0}; CY = (IY+d)[7]
SLA r	11001011	00100-r-			reg	3	7 (3,3,1)	*	*	0	P	0	*	r = {r[6,0],0}; CY = r[7]
SLP	11101101	01110110			none	3	8 (3,3,2)	-	-	-	-	-	-	Sleep
SRA (HL)	11001011	00101110			reg ind	5	13 (3,3,3,1,3)	*	*	0	P	0	*	(HL) = {(HL)[7],(HL)[7,1]}; CY = (HL)[0]
SRA (IX+d)	11011101	11001011	----d---	00101110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IX+d) = {(IX+d)[7],(IX+d)[7,1]}; CY = (IX+d)[0]
SRA (IY+d)	11111101	11001011	----d---	00101110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IY+d) = {(IY+d)[7],(IY+d)[7,1]}; CY = (IY+d)[0]
SRA r	11001011	00101-r-			reg	3	7 (3,3,1)	*	*	0	P	0	*	r = {r[7],r[7,1]}; CY = r[0]
SRL (HL)	11001011	00111110			reg ind	5	13 (3,3,3,1,3)	*	*	0	P	0	*	(HL) = {0,(HL)[7,1]}; CY = (HL)[0]
SRL (IX+d)	11011101	11001011	----d---	00111110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IX+d) = {0,(IX+d)[7,1]}; CY = (IX+d)[0]
SRL (IY+d)	11111101	11001011	----d---	00111110	index	7	19 (3,3,3,3,3,1,3)	*	*	0	P	0	*	(IY+d) = {0,(IY+d)[7,1]}; CY = (IY+d)[0]
SRL r	11001011	00111-r-			reg	3	7 (3,3,1)	*	*	0	P	0	*	r = {0,r[7,1]}; CY = r[0]
SUB (HL)	10010110				reg ind	2	6 (3,3)	*	*	*	V	1	*	A = A - (HL)
SUB (IX+d)	11011101	10010110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	1	*	A = A - (IX+d)
SUB (IY+d)	11111101	10010110	----d---		index	5	14 (3,3,3,2,3)	*	*	*	V	1	*	A = A - (IY+d)
SUB n	11010110	----n---			immed	2	6 (3,3)	*	*	*	V	1	*	A = A - n
SUB r	10010-r-				reg	2	4 (3,1)	*	*	*	V	1	*	A = A - r
TST (HL)	11101101	00110100			reg ind	4	10 (3,3,1,3)	*	:*	1	P	0	0	A & (HL)
TST n	11101101	01100100	----n---		immed	3	9 (3,3,3)	*	*	1	P	0	0	A & n
TST r	11101101	00-r-100			reg	3	7 (3,3,1)	*	*	1	P	0	0	A & r
TSTIO n	11101101	01110100	----n---		direct	4	12 (3,3,3,3)	*	*	1	P	0	0	(C) & n
XOR (HL)	10101110				reg ind	2	6 (3,3)	*	*	0	P	0	0	A = [A & ~(HL)] | [~A & (HL)]
XOR (IX+d)	11011101	10101110	----d---		index	5	14 (3,3,3,2,3)	*	*	0	P	0	0	A = [A & ~(IX+d)] | [~A & (IX+d)]
XOR (IY+d)	11111101	10101110	----d---		index	5	14 (3,3,3,2,3)	*	*	0	P	0	0	A = [A & ~(IY+d)] | [~A & (IY+d)]
XOR n	11101110	----n---			immed	2	6 (3,3)	*	*	0	P	0	0	A = [A & ~n] | [~A & n]
XOR r	10101-r-				reg	2	4 (3,1)	*	*	0	P	0	0	A = [A & ~r] | [~A & r]
ZIACK0					none	1	3	-	-	-	-	-	-	IEF1=0; IEF2=0
ZIACK1					none	3	9 (3,3,3)	-	-	-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP-2; IEF2 = 0; IEF1 = 0; PC = 0038h
ZIACK2					none	6	16 (3,1,3,3,3,3)	-	-	-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP-2; IEF2 = 0; IEF1 = 0; PC = (VT)
ZNMIACK					none	4	11 (3,2,3,3)	-	-	-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP-2; IEF2 = IEF1; IEF1 = 0; PC = 0066h
ZTRAP2					none	3	12 (6,3,3)	-	-	-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP-2; PC = 0000h
ZTRAP3					none	3	10 (4,3,3)	-	-	-	-	-	-	(SP-1) = PCH; (SP-2) = PCL; SP = SP-2; PC = 0000h

3.5.3 Address Bus Contents

The execution table below shows the contents of the Address Bus for each machine cycle. The Address Bus is only valid during memory, I/O, and interrupt acknowledge cycles, and will be undefined during internal operation cycles. The default address output is the Program Counter, so this is what will usually be on the Address Bus during internal operation cycles.

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
ADC A,(HL)	PC											HL				
ADC A,(IX+d)	PC		PC	PC			xx					IX+d				
ADC A,(IY+d)	PC		PC	PC			xx					IY+d				
ADC A,n	PC			PC												
ADC A,r	PC						xx									
ADC HL,ss	PC		PC				xx									
ADD A,(HL)	PC											HL				
ADD A,(IX+d)	PC		PC	PC			xx					IX+d				
ADD A,(IY+d)	PC		PC	PC			xx					IY+d				
ADD A,n	PC			PC												
ADD A,r	PC						xx									
ADD HL,ss	PC						xx									
ADD IX,xx	PC		PC				xx									
ADD IY,yy	PC		PC				xx									
AND (HL)	PC											HL				
AND (IX+d)	PC		PC	PC			xx					IX+d				
AND (IY+d)	PC		PC	PC			xx					IY+d				
AND n	PC			PC												
AND r	PC						xx									
BIT b,(HL)	PC		PC									HL				
BIT b,(IX+d))	PC		PC	PC		PC						IX+d				
BIT b,(IY+d))	PC		PC	PC		PC						IY+d				
BIT b,r	PC		PC													
CALL f,mn	PC			PC	PC		xx							SP-1	SP-2	
CALL mn	PC			PC	PC		xx							SP-1	SP-2	
CCF	PC															
CP (HL)	PC											HL				
CP (IX+d)	PC		PC	PC			xx					IX+d				
CP (IY+d)	PC		PC	PC			xx					IY+d				
CP n	PC			PC												
CP r	PC						xx									
CPD	PC		PC									HL	xx			
CPDR	PC		PC									HL	xx			xx
CPI	PC		PC									HL	xx			
CPIR	PC		PC									HL	xx			xx
CPL	PC															
DAA	PC						xx									
DEC (HL)	PC											HL	xx		HL	
DEC (IX+d)	PC		PC	PC			xx					IX+d	xx		IX+d	
DEC (IY+d)	PC		PC	PC			xx					IY+d	xx		IY+d	

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
DEC IX	PC		PC				xx									
DEC IY	PC		PC				xx									
DEC r	PC						xx									
DEC ss	PC						xx									
DI	PC															
DJNZ j	PC	xx		PC			xx									
EI	PC															
EX (SP),HL	PC										SP	SP+1	xx	SP+1	SP	
EX (SP),IX	PC		PC								SP	SP+1	xx	SP+1	SP	
EX (SP),IY	PC		PC								SP	SP+1	xx	SP+1	SP	
EX AF,AF'	PC						xx									
EX DE,HL	PC															
EXX	PC															
HALT	PC															
IM 0	PC		PC													
IM 1	PC		PC													
IM 2	PC		PC													
IN A,(n)	PC			PC								An				
IN r,(C)	PC		PC									BC				
IN0 r,(n)	PC		PC	PC								0n				
INC (HL)	PC											HL	xx		HL	
INC (IX+d)	PC		PC	PC			xx					IX+d	xx		IX+d	
INC (IY+d)	PC		PC	PC			xx					IY+d	xx		IY+d	
INC IX	PC		PC				xx									
INC IY	PC		PC				xx									
INC r	PC						xx									
INC ss	PC						xx									
IND	PC		PC									BC			HL	
INDR	PC		PC									BC			HL	xx
INI	PC		PC									BC			HL	
INIR	PC		PC									BC			HL	xx
JP (HL)	PC															
JP (IX)	PC		PC													
JP (IY)	PC		PC													
JP f,mn	PC			PC	PC											
JP mn	PC			PC	PC											
JR cc,j	PC			PC			xx									
JR j	PC			PC			xx									
LD (BC),A	PC						xx								BC	
LD (DE),A	PC						xx								DE	
LD (HL),n	PC			PC											HL	
LD (HL),r	PC						xx								HL	
LD (IX+d),n	PC		PC	PC	PC										IX+d	
LD (IX+d),r	PC		PC	PC			xx								IX+d	
LD (IY+d),n	PC		PC	PC	PC										IY+d	
LD (IY+d),r	PC		PC	PC			xx								IY+D	
LD (mn),A	PC			PC	PC		xx								mn	
LD (mn),HL	PC			PC	PC		xx							mn	mn+1	
LD (mn),IX	PC		PC	PC	PC		xx							mn	mn+1	
LD (mn),IY	PC		PC	PC	PC		xx							mn	mn+1	
LD (mn),ss	PC		PC	PC	PC		xx							mn	mn+1	

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
LD A,(BC)	PC											BC				
LD A,(DE)	PC											DE				
LD A,(mn)	PC			PC	PC							mn				
LD A,I	PC		PC													
LD A,R	PC		PC													
LD dd,(mn)	PC		PC	PC	PC						mn	mn+1				
LD dd,mn	PC			PC	PC											
LD HL,(mn)	PC			PC	PC						mn	mn+1				
LD I,A	PC		PC													
LD IX,(mn)	PC		PC	PC	PC						mn	mn+1				
LD IX,mn	PC		PC	PC	PC											
LD IY,(mn)	PC		PC	PC	PC						mn	mn+1				
LD IY,mn	PC		PC	PC	PC											
LD r,(HL)	PC											HL				
LD r,(IX+d)	PC		PC	PC			xx					IX+d				
LD r,(IY+d)	PC		PC	PC			xx					IY+d				
LD R,A	PC		PC													
LD r,n	PC			PC												
LD r,r'	PC						xx									
LD SP,HL	PC						xx									
LD SP,IX	PC		PC				xx									
LD SP,IY	PC		PC				xx									
LDD	PC		PC									HL			DE	
LDDR	PC		PC									HL			DE	xx
LDI	PC		PC									HL			DE	
LDIR	PC		PC									HL			DE	xx
MLT ww	PC		PC				xx									
NEG	PC		PC													
NOP	PC															
OR (HL)	PC											HL				
OR (IX+d)	PC		PC	PC			xx					IX+d				
OR (IY+d)	PC		PC	PC			xx					IY+d				
OR n	PC			PC												
OR r	PC						xx									
OTDM	PC		PC				xx					HL			0C	xx
OTDMR	PC		PC				xx					HL	xx		0C	xx
OTDR	PC		PC									HL			BC	xx
OTIM	PC		PC				xx					HL			0C	xx
OTIMR	PC		PC				xx					HL	xx		0C	xx
OTIR	PC		PC									HL			BC	xx
OUT (C),r	PC		PC				xx								BC	
OUT (n),A	PC			PC			xx								An	
OUT0 (n),r	PC		PC	PC			xx								0n	
OUTD	PC		PC									HL			BC	
OUTI	PC		PC									HL			BC	
POP IX	PC		PC								SP	SP+1				
POP IY	PC		PC								SP	SP+1				
POP zz	PC										SP	SP+1				
PUSH IX	PC		PC				xx							SP-1	SP-2	
PUSH IY	PC		PC				xx							SP-1	SP-2	
PUSH zz	PC						xx							SP-1	SP-2	

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
RES b,(HL)	PC		PC									HL	xx		HL	
RES b,(IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
RES b,(IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
RES b,r	PC		PC				xx									
RET	PC										SP	SP+1				
RET f	PC						xx				SP	SP+1	xx			
RETI	PC		PC				xx	PC	xx	PC	SP	SP+1				
RETN	PC		PC								SP	SP+1				
RL (HL)	PC		PC									HL	xx		HL	
RL (IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
RL (IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
RL r	PC		PC				xx									
RLA	PC															
RLC (HL)	PC		PC									HL	xx		HL	
RLC (IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
RLC (IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
RLC r	PC		PC				xx									
RLCA	PC															
RLD	PC		PC									HL	xx		HL	
RR (HL)	PC		PC									HL	xx		HL	
RR (IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
RR (IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
RR r	PC		PC				xx									
RRA	PC															
RRC (HL)	PC		PC									HL	xx		HL	
RRC (IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
RRC (IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
RRC r	PC		PC				xx									
RRCA	PC															
RRD	PC		PC									HL	xx		HL	
RST v	PC						xx							SP-1	SP-2	
SBC (IX+d)	PC		PC	PC			xx					HL				
SBC (IY+d)	PC		PC	PC			xx					IX+d				
SBC A,(HL)	PC											IY+d				
SBC A,n	PC			PC												
SBC A,r	PC						xx									
SBC HL,ss	PC		PC				xx									
SCF	PC															
SET b,(HL)	PC		PC									HL	xx		HL	
SET b,(IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
SET b,(IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
SET b,r	PC		PC				xx									
SLA (HL)	PC		PC									HL	xx		HL	
SLA (IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
SLA (IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
SLA r	PC		PC				xx									
SLP	PC		PC				xx									
SRA (HL)	PC		PC									HL	xx		HL	
SRA (IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
SRA (IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
SRA r	PC		PC				xx									

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
SRL (HL)	PC		PC									HL	xx		HL	
SRL (IX+d)	PC		PC	PC		PC						IX+d	xx		IX+d	
SRL (IY+d)	PC		PC	PC		PC						IY+d	xx		IY+d	
SRL r	PC		PC				xx									
SUB (HL)	PC											HL				
SUB (IX+d)	PC		PC	PC			xx					IX+d				
SUB (IY+d)	PC		PC	PC			xx					IY+d				
SUB n	PC			PC												
SUB r	PC						xx									
TST (HL)	PC		PC				xx					HL				
TST n	PC		PC	PC												
TST r	PC		PC				xx									
TSTIO n	PC		PC	PC								0C				
XOR (HL)	PC											HL				
XOR (IX+d)	PC		PC	PC			xx					IX+d				
XOR (IY+d)	PC		PC	PC			xx					IY+d				
XOR n	PC			PC												
XOR r	PC						xx									
ZIACK0																
ZIACK1														SP-1	SP-2	
ZIACK2											VT	VT+1	xx	SP-1	SP-2	
ZNMIACK													xx	SP-1	SP-2	
ZTRAP2													xx	SP-1	SP-2	
ZTRAP3													xx	SP-1	SP-2	

3.5.4 Next Machine State

The execution table below shows the sequence of machine cycles for each instruction or exception condition. All instructions start with the IF1 (instruction fetch 1) state, while exception conditions start with some kind of interrupt acknowledge state or the instruction fetch where the illegal opcode was fetched, which are not shown in the table. In each column is listed the next machine cycle for each instruction or exception. The word "done" in a column means that the corresponding machine cycle is the last one for that particular instruction or exception condition, and the next state will be either IF1 (for execution of another instruction) or an interrupt acknowledge cycle if an interrupt condition is present. Where there are two entries listed in a column for an instruction, the next state depends on the condition being tested in the instruction. The top entry corresponds to the condition being false, while the bottom entry corresponds to the condition being true. Shaded entries are not used for that particular instruction or exception condition. The names and descriptions of the machine cycles are listed following the table.

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
ADC A,(HL)	rd2											done				
ADC A,(IX+d)	if2		of1	iop2			rd2					done				
ADC A,(IY+d)	if2		of1	iop2			rd2					done				
ADC A,n	of1			done												
ADC A,r	iop1						done									
ADC HL,ss	if2		iop4				done									
ADD A,(HL)	rd2											done				
ADD A,(IX+d)	if2		of1	iop2			rd2					done				
ADD A,(IY+d)	if2		of1	iop2			rd2					done				
ADD A,n	of1			done												
ADD A,r	iop1						done									
ADD HL,ss	iop4						done									
ADD IX,xx	if2		iop4				done									
ADD IY,yy	if2		iop4				done									
AND (HL)	rd2											done				
AND (IX+d)	if2		of1	iop2			rd2					done				
AND (IY+d)	if2		of1	iop2			rd2					done				
AND n	of1			done												
AND r	iop1						done									
BIT b,(HL)	if2		rd2									done				
BIT b,(IX+d))	if2		of1	if3		rd2						done				
BIT b,(IY+d))	if2		of1	if3		rd2						done				
BIT b,r	if2		done													
CALL f,mn	of1			done of2	iop1		wr1							wr2	done	
CALL mn	of1			of2	iop1		wr1							wr2	done	
CCF	done															
CP (HL)	rd2											done				
CP (IX+d)	if2		of1	iop2			rd2					done				
CP (IY+d)	if2		of1	iop2			rd2					done				
CP n	of1			done												
CP r	iop1						done									

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
CPD	if2		rd2									siop3	done			
CPDR	if2		rd2									siop3	done fiop2			done
CPI	if2		rd2									siop3	done			
CPIR	if2		rd2									siop3	done fiop2			done
CPL	done															
DAA	iop1						done									
DEC (HL)	rd2											siop1	wr2		done	
DEC (IX+d)	if2		of1	iop2			rd2					siop1	wr2		done	
DEC (IY+d)	if2		of1	iop2			rd2					siop1	wr2		done	
DEC IX	if2		iop1				done									
DEC IY	if2		iop1				done									
DEC r	iop1						done									
DEC ss	iop1						done									
DI	done															
DJNZ j	dly	of1		done iop2			done									
EI	done															
EX (SP),HL	rd1										rd2	siop1	wr1	wr2	done	
EX (SP),IX	if2		rd1								rd2	siop1	wr1	wr2	done	
EX (SP),IY	if2		rd1								rd2	siop1	wr1	wr2	done	
EX AF,AF'	iop1						done									
EX DE,HL	done															
EXX	done															
HALT	done															
IM 0	if2		done													
IM 1	if2		done													
IM 2	if2		done													
IN A,(n)	of1			rd2								done				
IN r,(C)	if2		rd2									done				
IN0 r,(n)	if2		of1	rd2								done				
INC (HL)	rd2											siop1	wr2		done	
INC (IX+d)	if2		of1	iop2			rd2					siop1	wr2		done	
INC (IY+d)	if2		of1	iop2			rd2					siop1	wr2		done	
INC IX	if2		iop1				done									
INC IY	if2		iop1				done									
INC r	iop1						done									
INC ss	iop1						done									
IND	if2		rd2									wr2			done	
INDR	if2		rd2									wr2			done fiop2	done
INI	if2		rd2									wr2			done	
INIR	if2		rd2									wr2			done fiop2	done
JP (HL)	done															
JP (IX)	if2		done													
JP (IY)	if2		done													
JP f,mn	of1			done of2	done											
JP mn	of1			of2	done											
JR cc,j	of1			done iop2			done									
JR j	of1			iop2			done									
LD (BC),A	iop1						wr2								done	
LD (DE),A	iop1						wr2								done	
LD (HL),n	of1			wr2											done	
LD (HL),r	iop1						wr2								done	

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
LD (IX+d),n	if2		of1	of2	wr2										done	
LD (IX+d),r	if2		of1	iop3			wr2								done	
LD (IY+d),n	if2		of1	of2	wr2										done	
LD (IY+d),r	if2		of1	iop3			wr2								done	
LD (mn),A	of1			of2	iop1		wr2								done	
LD (mn),HL	of1			of2	iop1		wr1							wr2	done	
LD (mn),IX	if2		of1	of2	iop1		wr1							wr2	done	
LD (mn),IY	if2		of1	of2	iop1		wr1							wr2	done	
LD (mn),ss	if2		of1	of2	iop1		wr1							wr2	done	
LD A,(BC)	rd2											done				
LD A,(DE)	rd2											done				
LD A,(mn)	of1			of2	rd2							done				
LD A,I	if2		done													
LD A,R	if2		done													
LD dd,(mn)	if2		of1	of2	rd1						rd2	done				
LD dd,mn	of1			of2	done											
LD HL,(mn)	of1			of2	rd1						rd2	done				
LD I,A	if2		done													
LD IX,(mn)	if2		of1	of2	rd1						rd2	done				
LD IX,mn	if2		of1	of2	done											
LD IY,(mn)	if2		of1	of2	rd1						rd2	done				
LD IY,mn	if2		of1	of2	done											
LD r,(HL)	rd2											done				
LD r,(IX+d)	if2		of1	iop2			rd2					done				
LD r,(IY+d)	if2		of1	iop2			rd2					done				
LD R,A	if2		done													
LD r,n	of1			done												
LD r,r'	iop1						done									
LD SP,HL	iop1						done									
LD SP,IX	if2		iop1				done									
LD SP,IY	if2		iop1				done									
LDD	if2		rd2									wr2			done	
LDDR	if2		rd2									wr2			done fiop2	done
LDI	if2		rd2									wr2			done	
LDIR	if2		rd2									wr2			done fiop2	done
MLT ww	if2		iop10				done									
NEG	if2		done													
NOP	done															
OR (HL)	rd2											done				
OR (IX+d)	if2		of1	iop2			rd2					done				
OR (IY+d)	if2		of1	iop2			rd2					done				
OR n	of1			done												
OR r	iop1						done									
OTDM	if2		iop1				rd2					wr2			fiop1	done
OTDMR	if2		iop1				rd2					wr2			fiop1 fiop3	done
OTDR	if2		rd2									wr2			done fiop2	done
OTIM	if2		iop1				rd2					wr2			fiop1	done
OTIMR	if2		iop1				rd2					wr2			fiop1 fiop3	done
OTIR	if2		rd2									wr2			done fiop2	done
OUT (C),r	if2		iop1				wr2								done	
OUT (n),A	of1			iop1			wr2								done	
OUT0 (n),r	if2		of1	iop1			wr2								done	

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
OUTD	if2		rd2									wr2			done	
OUTI	if2		rd2									wr2			done	
POP IX	if2		rd1								rd2	done				
POP IY	if2		rd1								rd2	done				
POP zz	rd1										rd2	done				
PUSH IX	if2		iop2				wr1							wr2	done	
PUSH IY	if2		iop2				wr1							wr2	done	
PUSH zz	iop2						wr1							wr2	done	
RES b,(HL)	if2		rd2									siop1	wr2		done	
RES b,(IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
RES b,(IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
RES b,r	if2		iop1				done									
RET	rd1										rd2	done				
RET f	iop2 iop1						done rd1				rd2	done				
RETI	if2		rd1 iop3				sif1	sdly	sif2	rd1	rd2	done				
RETN	if2		rd1								rd2	done				
RL (HL)	if2		rd2									siop1	wr2		done	
RL (IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
RL (IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
RL r	if2		iop1				done									
RLA	done															
RLC (HL)	if2		rd2									siop1	wr2		done	
RLC (IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
RLC (IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
RLC r	if2		iop1				done									
RLCA	done															
RLD	if2		rd2									siop4	wr2		done	
RR (HL)	if2		rd2									siop1	wr2		done	
RR (IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
RR (IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
RR r	if2		iop1				done									
RRA	done															
RRC (HL)	if2		rd2									siop1	wr2		done	
RRC (IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
RRC (IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
RRC r	if2		iop1				done									
RRCA	done															
RRD	if2		rd2									siop4	wr2		done	
RST v	iop2						wr1							wr2	done	
SBC (IX+d)	if2		of1	iop2			rd2					done				
SBC (IY+d)	if2		of1	iop2			rd2					done				
SBC A,(HL)	rd2											done				
SBC A,n	of1			done												
SBC A,r	iop1						done									
SBC HL,ss	if2		iop4				done									
SCF	done															
SET b,(HL)	if2		rd2									siop1	wr2		done	
SET b,(IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
SET b,(IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
SET b,r	if2		iop1				done									

Instruction	if1	dly	if2	of1	of2	if3	iop	sif1	sdly	sif2	rd1	rd2	siop	wr1	wr2	fiop
SLA (HL)	if2		rd2									siop1	wr2		done	
SLA (IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
SLA (IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
SLA r	if2		iop1				done									
SLP	if2		iop2				done									
SRA (HL)	if2		rd2									siop1	wr2		done	
SRA (IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
SRA (IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
SRA r	if2		iop1				done									
SRL (HL)	if2		rd2									siop1	wr2		done	
SRL (IX+d)	if2		of1	if3		rd2						siop1	wr2		done	
SRL (IY+d)	if2		of1	if3		rd2						siop1	wr2		done	
SRL r	if2		iop1				done									
SUB (HL)	rd2											done				
SUB (IX+d)	if2		of1	iop2			rd2					done				
SUB (IY+d)	if2		of1	iop2			rd2					done				
SUB n	of1			done												
SUB r	iop1						done									
TST (HL)	if2		iop1				rd2					done				
TST n	if2		of1	done												
TST r	if2		iop1				done									
TSTIO n	if2		of1	rd2								done				
XOR (HL)	rd2											done				
XOR (IX+d)	if2		of1	iop2			rd2					done				
XOR (IY+d)	if2		of1	iop2			rd2					done				
XOR n	of1			done												
XOR r	iop1						done									
ZIACK0																
ZIACK1							wr1							wr2	done	
ZIACK2							wr1				rd2	done		wr2	done	
ZNMIACK							wr1							wr2	rd1	
ZTRAP2													wr1	wr2	done	
ZTRAP3													wr1	wr2	done	

if1, if2 and if3 are instruction fetch cycles, for the first, second and third opcode respectively.

dly is used only with DJNZ to speed operation by decrementing and checking b before fetching the displacement.

of1 and of2 are the operand fetch cycles.

iop are internal operation cycles and can be up to 10 clocks long.

sif1 and sif2 are the RETI instruction refetch cycles.

sdly is a delay cycle needed between the RETI instruction refetch cycles.

rd1 and rd2 are memory or I/O read cycles. rd2 is used for byte reads, and both rd1 and rd2 are used for word reads.

siop are internal operation cycles and can be up to 5 clocks.

wr1 and wr2 are memory or I/O write cycles. wr2 is used for byte writes, and both wr1 and wr2 are used for word writes.

fiop are internal operation cycles and can be up to 3 clocks long.

4 Pin Descriptions

This section describes the pins of the Y180 model. All input pins are sampled by CLK_, CLKB_, or both. All output pins come from flip-flops, although if a pin changes on both edges of CLK_, the pin will be a simple combination of two flip-flop outputs. The table below shows pin names, direction, function and sampling or changing CLK_ edge.

Pin name	Direction	Function	Changes on/ Sampled on
A_[15:0]	Output	Address Bus	CLK_
AOEB_	Output	Address Output Enable	CLK_
BUSACKB_	Output	Bus Acknowledge	Both
BUSREQB_	Input	Bus Request	CLKB_
CLEARB_	Input	Master Clear	CLKB_
CLK_	Input	Clock	
CLKB_	Input	Clock-Bar	
COEB_	Output	Control Output Enable	CLK_
DIN_[7:0]	Input	Data Input Bus	Both
DOEB_	Output	Data Output Enable	CLK_
DOUT_[7:0]	Output	Data Output Bus	CLK_
E_	Output	Enable	Both
HALTB_	Output	Halt Mode	CLK_
INTB_	Input	Interrupt Request	CLKB_
IOCB_	Input	I/O Control Select	CLK_
IORQB_	Output	I/O Request	Both
M1B_	Output	Machine Cycle 1	CLK_
M1E_	Input	Machine Cycle 1 Enable	CLK_
MREQB_	Output	Memory Request	Both
NMIB_	Input	Non-Maskable Interrupt Request	CLKB_
RDB_	Output	Read	Both
RESETB_	Input	Master Reset	CLKB_
SLPB_	Output	Sleep Mode	CLK_
ST_	Output	Strobe	CLK_
TRAPB_	Output	Trap	CLK_
WAITB_	Input	Wait Request	CLKB_
WRB_	Output	Write	Both

4.1 A_[15:0] (Address Bus)

The 16 bit Address Bus is used to address memory and I/O. The address on this bus will be valid throughout a memory or I/O cycle, but the contents are undefined during internal operation cycles. The default address output is the Program Counter, so this is what will usually be on the A_[15:0] during internal operation cycles.

4.2 AOEB_ (Address Output Enable)

The Address Output Enable signal can be used to control 3-state buffers on the address bus external to the Y180. This signal will be active (Low) when the Y180 should be driving the address bus and inactive (High) when the Y180 is releasing the bus for another bus master to drive the address bus.

4.3 BUSACKB_ (Bus Acknowledge)

The Bus Acknowledge signal is active (Low) when the Y180 has relinquished control of the address bus, data bus and control signals to another bus master in response to a request on the BUSREQB_ signal.

4.4 BUSREQB_ (Bus Request)

When the Bus Request signal is active (Low), the Y180 will relinquish control of the address bus, data bus and control signals upon completion of the current machine cycle and then signal that it has done so by activating the BUSACKB_ signal. An external bus master may then take control of these buses. The BUSREQB_ is the highest priority request (except for RESETB_) that will be accepted by the Y180. BUSREQB_ cannot be masked, and is higher priority than NMIB_.

4.5 CLEARB_ (Master Clear)

The Master Clear signal should be activated (Low) on power-up at the same time as the RESETB_ signal, but only if the contents of the register file need to be initialized to known values. CLEARB_ will reset all register file contents to all zeros, as opposed to RESETB_, which only initializes a few registers. Do not active CLEARB_ at any other time, unless you really want to clear the register file. In particular, if you are exiting Halt mode or Sleep mode with reset, use RESETB_ only, unless register file data does not need to be preserved.

4.6 CLK_ (Clock)

This is the master Clock input. All internal signals change state on the rising edge of this clock, as it goes to the clock input of all internal flip-flops. A separate CLKB_ input is present on the Y180 to allow for compatibility with the Z180 timing on some inputs as well as some outputs. But CLK_ is used exclusively for internal flip-flops. Care should be exercised when routing CLK_ to minimize skew, and the buffer chosen must have sufficient drive for the load presented by all of these flip-flops. Timing analysis should always be performed after layout to verify proper operation.

4.7 CLKB_ (Clock-Bar)

This is the master Clock-Bar input. It is used only in the IO_CTRL module of the Y180 to provide compatible timing. CLKB_ is the inverse of CLK_, and is only lightly loaded. Only the rising edge of CLKB_ (which corresponds to the falling edge of CLK_) is ever used. The design of the CLK_ and CLKB_ buffers depends on the target technology for the Y180 and cannot be overemphasized.

4.8 COEB_ (Control Output Enable)

The Control Output Enable signal can be used to control 3-state buffers on the various control signals external to the Y180. This signal will be active (Low) when the Y180 should be driving these control signals and inactive (High) when the Y180 is releasing the bus for another bus master to drive these control signals. Typically, these control signals would consist of MREQB_, IORQB_, RDB_, and WRB_.

4.9 DIN_ [7:0] (Data Input Bus)

The 8 bit Data Input Bus is used to communicate data into the Y180. DIN_[7:0] is latched by the rising edge of CLK_ for instruction fetch cycles and for the interrupt acknowledge cycles in interrupt mode 0. In all other cases, the DIN_[7:0] is sampled by the rising edge of CLKB_. The DIN_[7:0] and the DOUT_[7:0] may be combined externally to the Y180, with the direction of this bus controlled by the DOEB_ signal.

4.10 DOEB_ (Data Output Enable)

The Data Output Enable signal can be used to control 3-state buffers on DOUT_[7:0] external to the Y180, or to control the 3-state buffers on a bidirectional data bus external to the Y180. This signal will be active (Low) when the Y180 should be

driving the data bus and inactive (High) when the Y180 is either reading data from the bus or releasing the bus for another bus master to drive.

4.11 DOUT_ [7:0] (Data Output Bus)

The 8 bit Data Output Bus is used to communicate data from the Y180. DOUT_[7:0] changes on the rising edge of CLK_ and is valid only for the duration of the write cycle.

4.12 E_ (Enable)

The Enable signal is a synchronous machine-cycle clock that is active (High) during bus transactions. It can be used by some peripheral families as a data strobe.

4.13 HALTB_ (Halt Mode)

The Halt Mode signal is active (Low) while the Y180 is in Halt mode or Sleep mode. Halt mode is entered when the HALT instruction is executed, while Sleep mode is entered when the SLP instruction is executed. In either case, the Y180 will remain in this mode until either a RESETB_, INTB_ or NMIB_ occurs.

4.14 INTB_ (Interrupt Request)

When the Interrupt Request input is active (Low) at the end of the current instruction, and neither BUSREQB_ or NMIB_ is active, the Y180 will perform an interrupt acknowledge cycle and go to the interrupt service routine. The particular interrupt acknowledge cycle depends on the interrupt mode of the Y180, and the request will be ignored if interrupts are not enabled in the Y180.

4.15 IOCB_ (I/O Control Select)

The I/O Control Select signal controls the timing of the IORQB_ and RDB_ signals during an I/O transaction. If the IOCB_ signal is Low, these two control signals go active (Low) during I/O transactions on the rising edge of CLK_. If the IOCB_ signal is High, these two control signals go active one half of a clock cycle earlier in the I/O transaction, on the rising edge of CLKB_. The trailing edge of these two control signals is not affected by the state of the signal.

4.16 IORQB_ (I/O Request)

The I/O Request signal is active (Low) during I/O cycles, and also during interrupt acknowledge cycles when the interrupt vector or instruction should be placed on the DIN_[7:0].

4.17 M1B_ (Machine Cycle 1)

The Machine Cycle 1 signal is active (Low) during instruction fetch cycles. It will be active during all instruction fetch cycles if the M1E_ signal is High, and only during the refetch of the RETI instruction if the M1E_ signal is Low. The M1B_ signal is always activated during interrupt acknowledge cycles.

4.18 M1E_ (Machine Cycle 1 Enable)

The Machine Cycle 1 Enable signal controls the operation of the M1B_ signal. If the M1E_ signal is High, the M1B_ signal will be activated for every instruction fetch. If the M1E_ signal is Low, the M1B_ signal will only be active during the refetch of the RETI instruction. This signal has no effect on the operation of the M1B_ signal during interrupt acknowledge cycles, where it is always active.

4.19 MREQB_ (Memory Request)

The Memory Request signal is active (Low) during memory cycles, and also during non-maskable interrupt acknowledge cycles.

4.20 NMIB_ (Non-Maskable Interrupt Request)

When the Non-Maskable Interrupt Request input is active (Low) for two successive rising edges of CLKB_, this information is latched and at the end of the current instruction the Y180 will perform a non-maskable interrupt acknowledge cycle and jump to location 0066h for the NMI service routine. The NMI service routine should be terminated with the RETN instruction for proper handling of the maskable interrupt. If BUSREQB_ is active concurrently with the NMIB_, BUSREQB_ will be given priority.

4.21 RDB_ (Read)

The Read signal is active (Low) during memory and I/O read cycles, and also during non-maskable interrupt acknowledge cycles.

4.22 RESETB_ (Master Reset)

The Master Reset signal should be activated (Low) on power-up and at any other time where initializing the Y180 to a known state is necessary. RESETB_ forces all output signals inactive, resets all internal state machines, clears the Program Counter, Stack Pointer, I register and R register. If the remaining registers need to be initialized to known states, the CLEARB_ signal should be simultaneously active.

4.23 SLPB_ (Sleep Mode)

The Sleep Mode signal is active (Low) while the Y180 is in Sleep mode. Sleep mode is entered when the SLP instruction is executed. The Y180 will remain in Sleep mode until either a RESETB_, INTB_ or NMIB_ occurs.

4.24 ST_ (Status)

The Status signal is used to aid in decoding of the current machine cycle, especially when the M1E_ has disabled the activation of the M1B_ signal. The ST_ signal is always active (Low) during the first instruction fetch cycle of an instruction, and also during the Halt Mode.

4.25 TRAPB_ (Trap)

The Trap signal is active (Low) for one clock cycle whenever the Y180 has encountered an undefined opcode. If TRAPB_ is active while the RDB_ signal is High, the undefined opcode occured in the second byte of the instruction, while if TRAPB_ is Active while the RDB_ signal is Low, the undefined opcode occured in the third byte of the instruction.

4.26 WAITB_ (Wait Request)

When the Wait Request input is active (Low) during a read, write, or interrupt acknowledge cycle, the cycle is extended for the duration of the WAITB_ Low time, one clock cycle at a time. The cycle then finishes when the WAITB_ signal returns High. This allows slow memory or peripheral device time to respond to bus cycles.

4.27 WRB_ (Write)

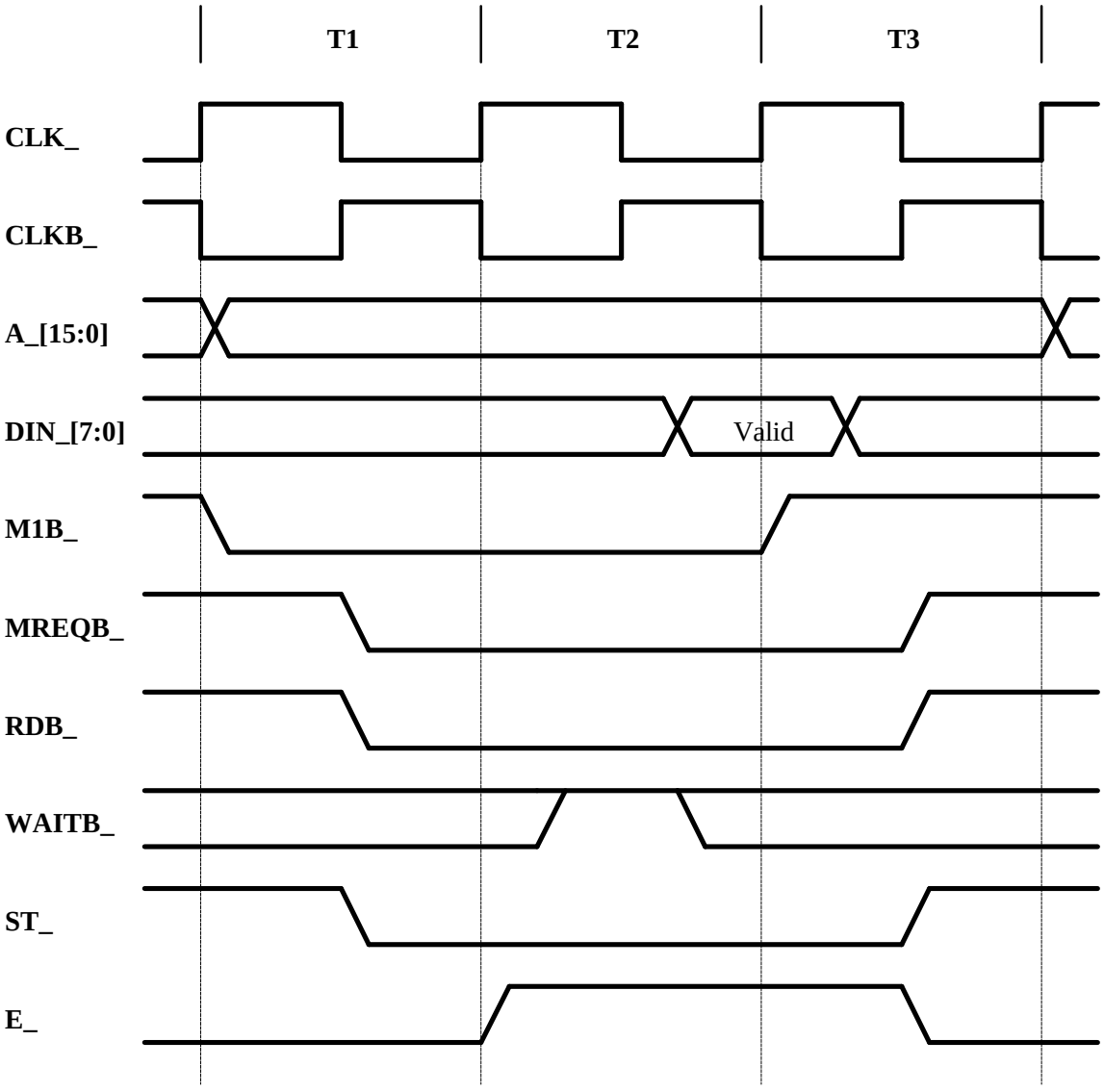
The Write signal is active (Low) during memory and I/O write cycles.

5 Bus Cycles

The figures below show the various bus cycles for the Y180. Throughout the figures, only the relevant pins are shown.

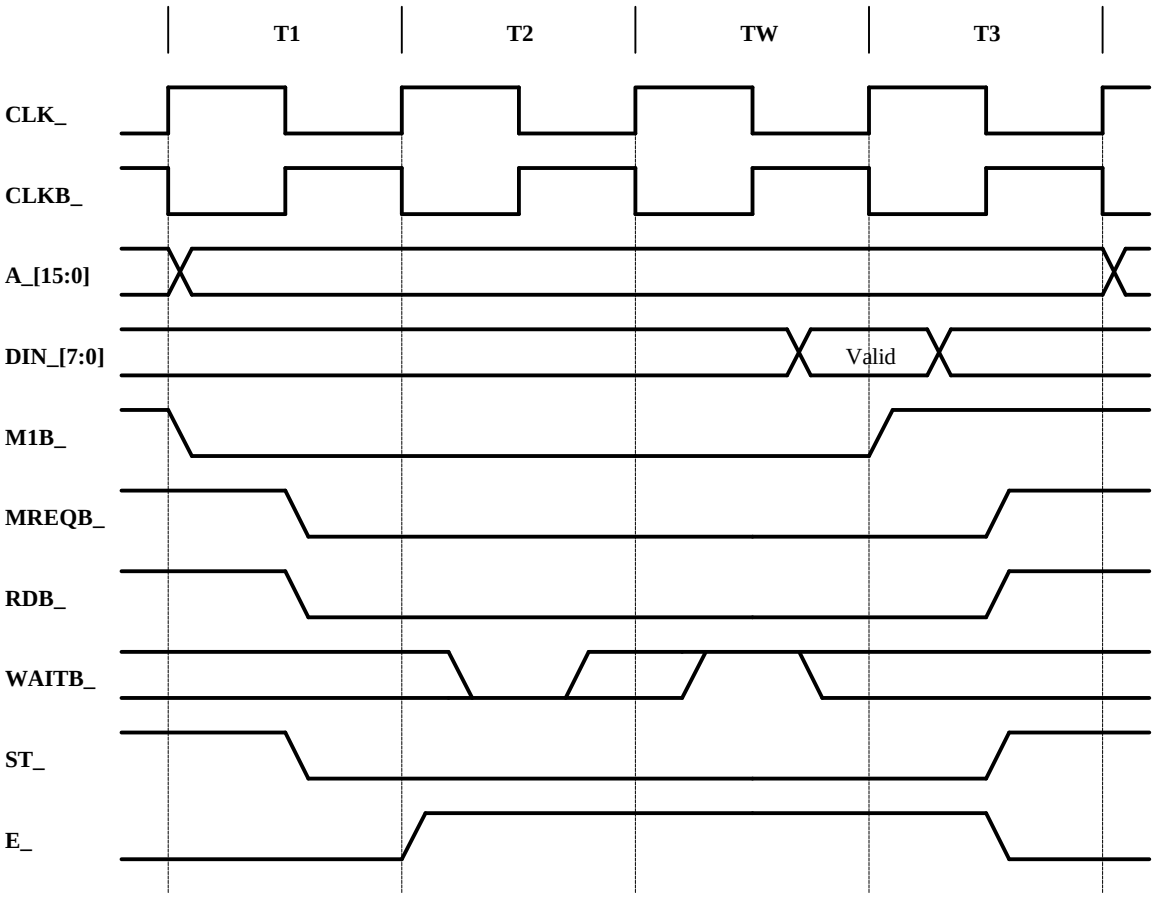
5.1 Instruction Fetch (without Wait state)

The timing for an instruction fetch cycle is shown below. This bus cycle is three clock cycles long, with the WAITB_ input sampled at the falling edge of CLK_ in T2, and the DIN_ bus sampled at the rising edge of CLK_ in T3. The ST_ signal is Low only for the fetch of the first byte of an instruction, and the M1B_ signal is asserted Low during instruction fetch cycles only if the M1E_ input is High.



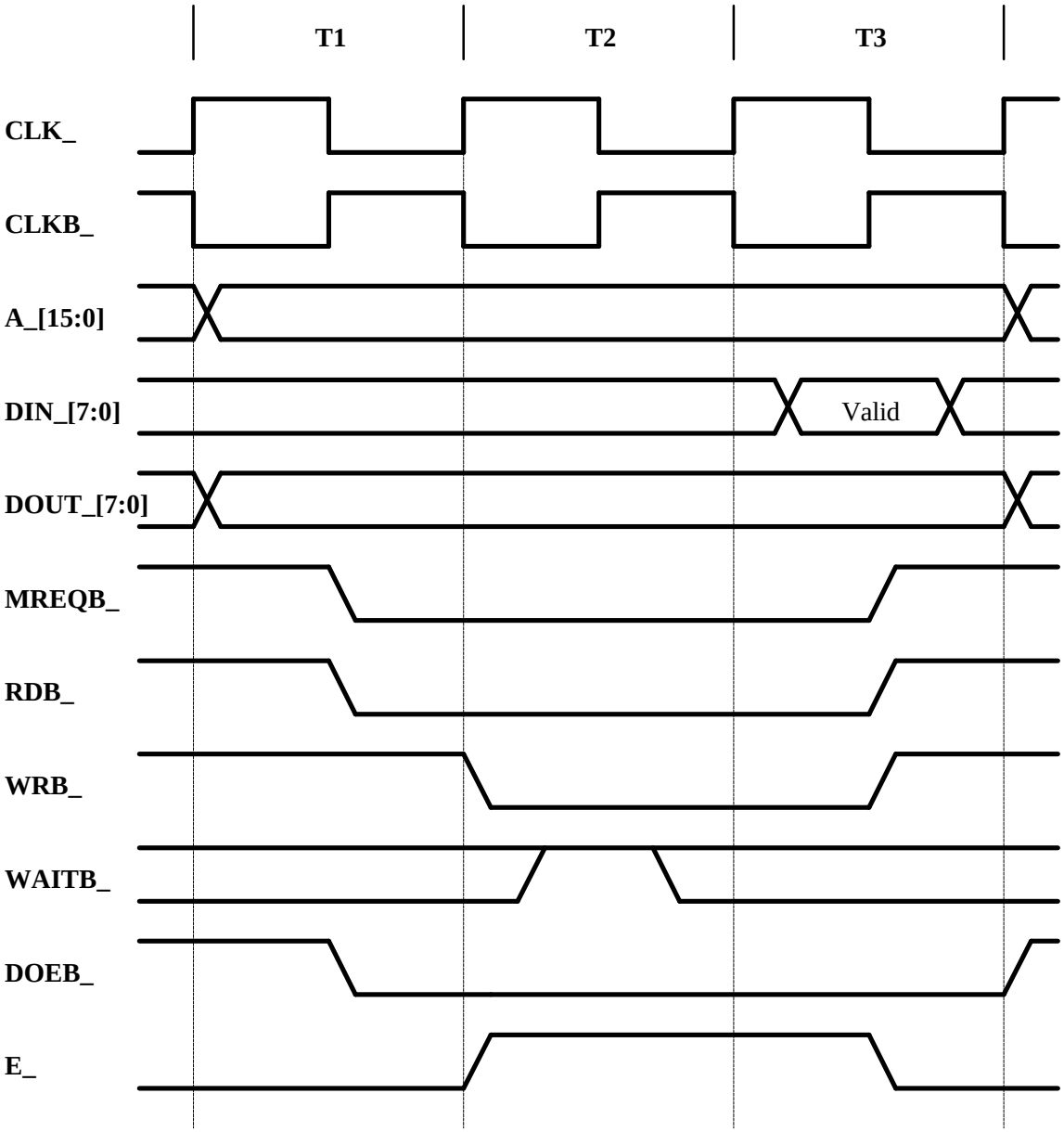
5.2 Instruction Fetch (with Wait state)

The timing for an instruction fetch cycle with one Wait state is shown below. This bus cycle is now four clock cycles long, with the WAITB_ input sampled at the falling edge of CLK_ in both T2 and TW, and the DIN_ bus sampled at the rising edge of T3. All of the control signals are stretched by the insertion of the Wait state.



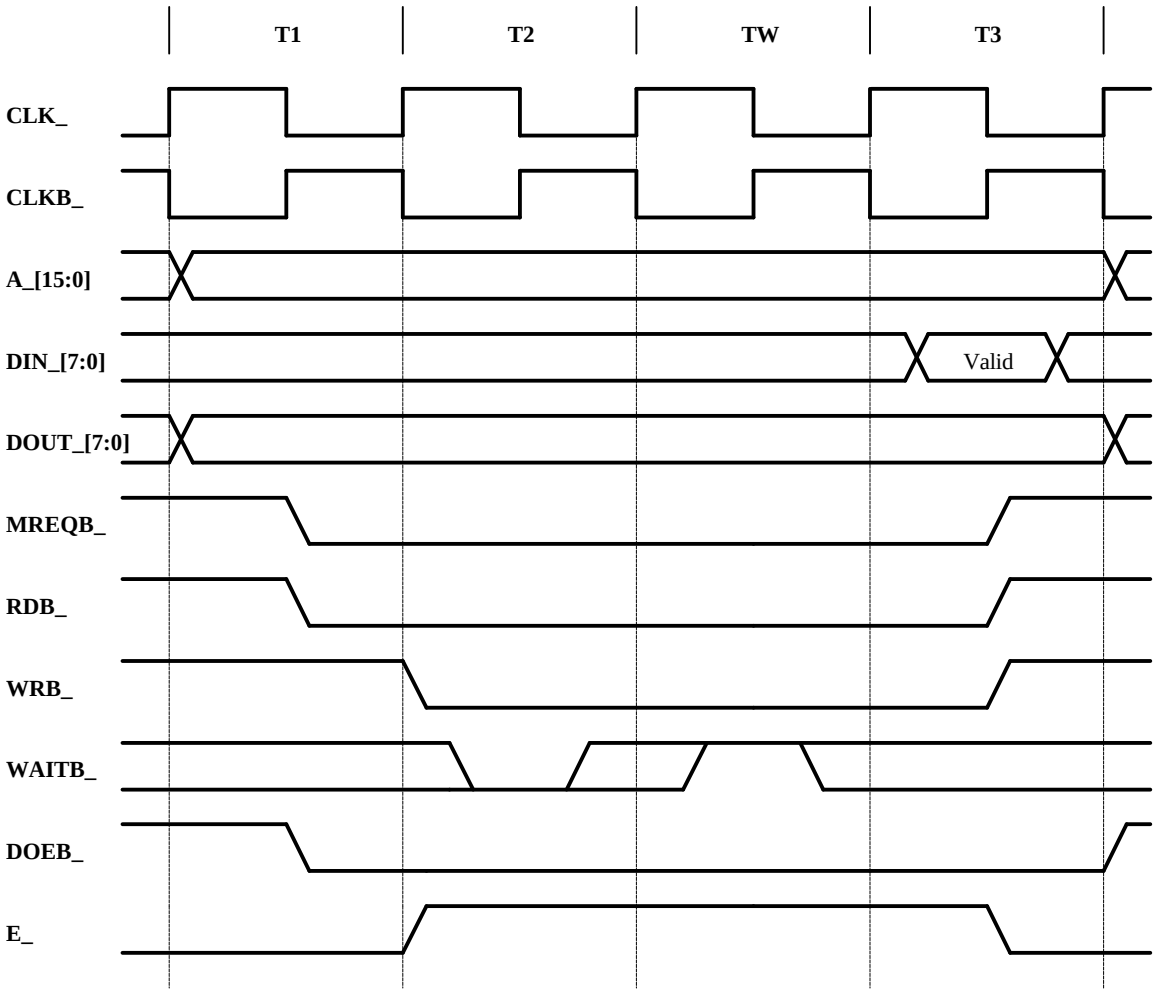
5.3 Memory Read/Write (without Wait State)

The timing for a memory read or memory write cycle is shown below. These bus cycles are three clock cycles long, with the WAITB_ input sampled at the falling edge of CLK_ in T2. In the case of memory read, the DIN_ bus sampled at the falling edge of CLK_ in T3 and the RDB_ signal is activated. In the case of memory write, the DOUT_ bus is driven with valid data for the duration of a memory write cycle and the WRB_ and DOEB_ signals are activated. The DOEB_ signal can be used to control buffer direction if a 3-state bus is used externally to the model.



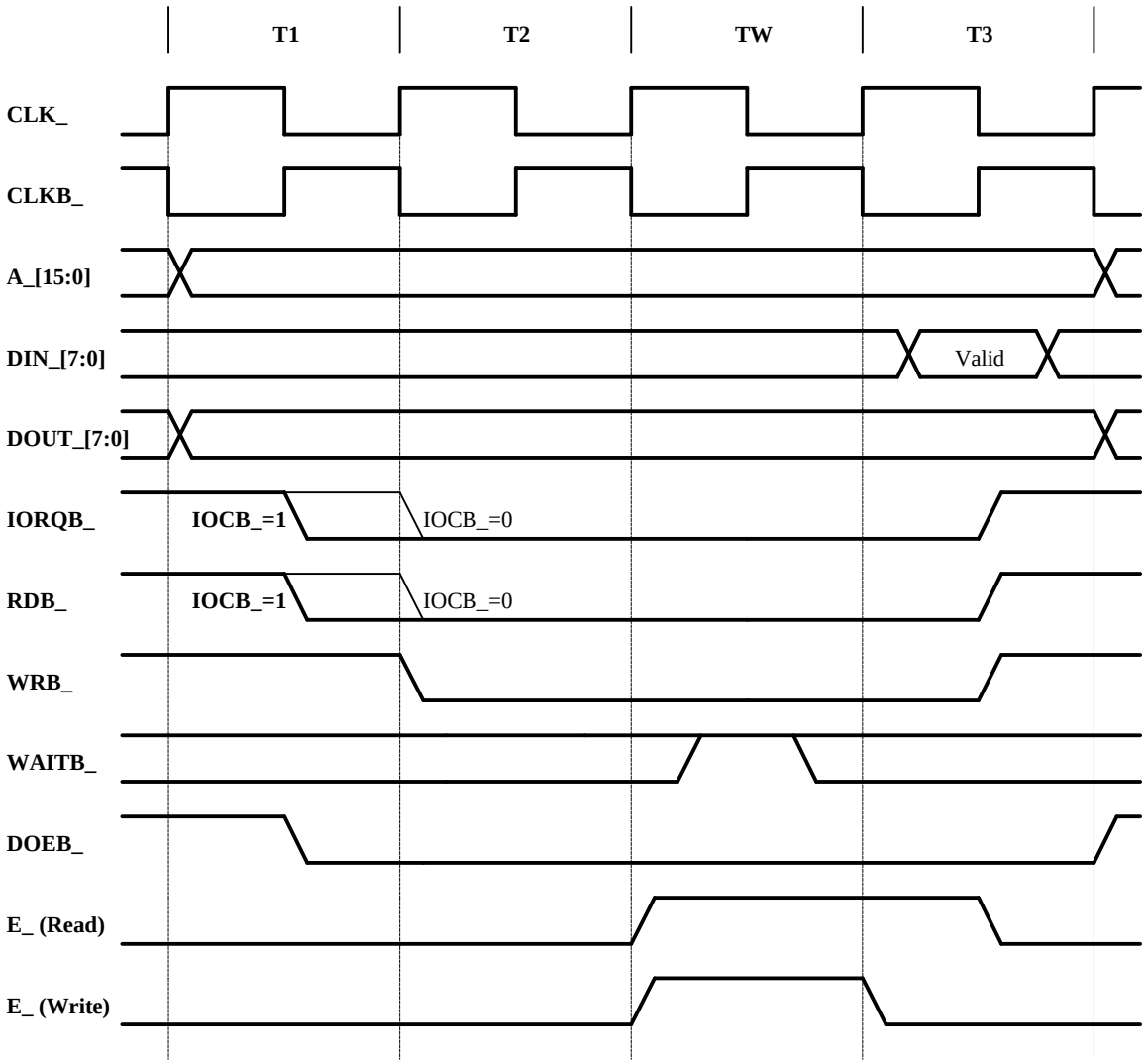
5.4 Memory Read/Write (with Wait State)

The timing for a memory read or memory write cycle with one Wait state is shown below. These bus cycles are now four clock cycles long, with the WAITB_ input sampled at the falling edge of CLK_ in both T2 and TW. All of the control signals are stretched by the insertion of the Wait state.



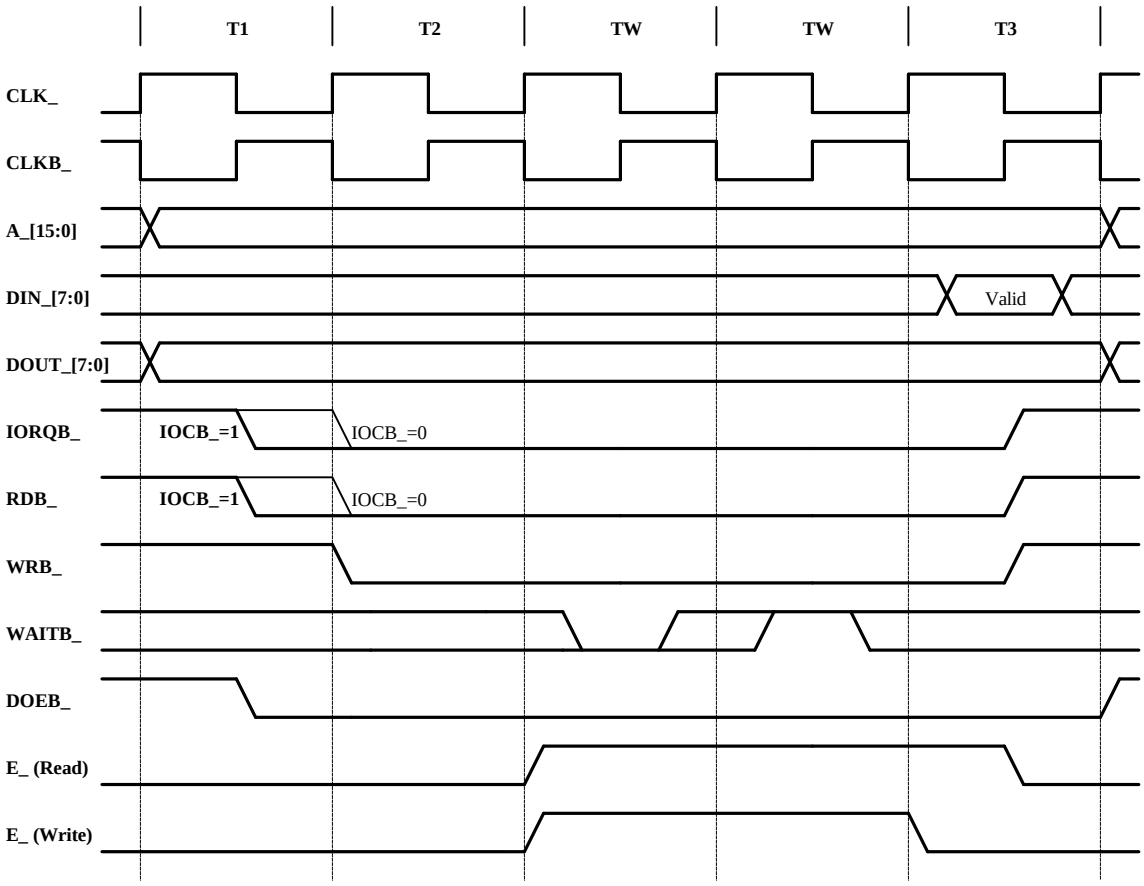
5.5 I/O Read/Write (without Wait State)

The timing for an I/O read or I/O write cycle is shown below. These bus cycles are four clock cycles long (three plus an automatic Wait state), with the WAITB_ input sampled at the falling edge of CLK_ in TW. In the case of I/O read, the DIN_ bus sampled at the falling edge of CLK_ in T3 and the RDB_ signal is activated. In the case of I/O write, the DOUT_ bus is driven with valid data for the duration of a I/O write cycle and the WRB_ and DOEB_ signals are activated. The IORQB_ signal is used to distinguish I/O read and write cycles from memory read and write cycles. Note that the timing of the leading edge of IORQB_ and RDB_ are controlled by the IOCB_ input. Also note that the timing of the E_ signal is different for I/O read and I/O write.



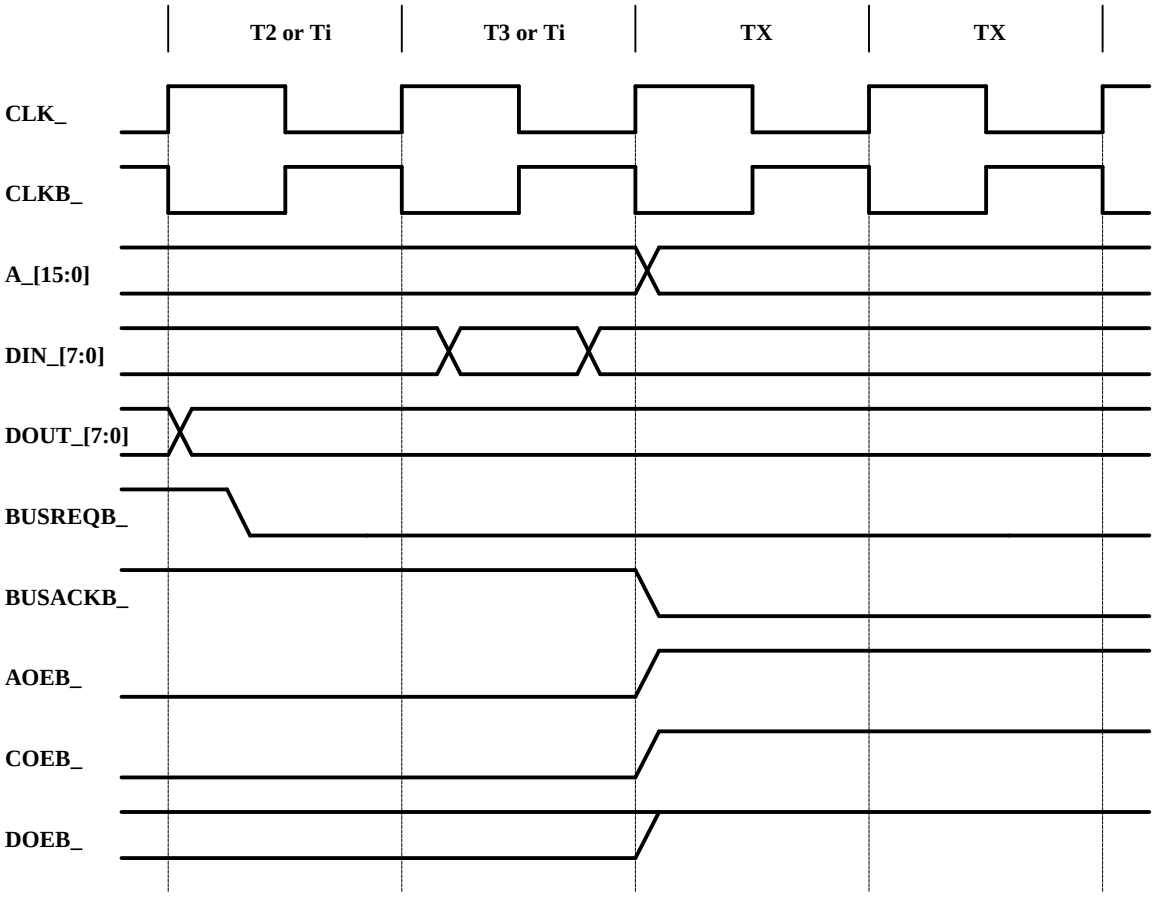
5.6 I/O Read/Write (with Wait State)

The timing for an I/O read or I/O write cycle with one inserted Wait state is shown below. These bus cycles are five clock cycles long (three plus one automatic and one inserted Wait state), with the WAITB_ input sampled at the falling edge of CLK_ in TW. All of the control signals are stretched by the insertion of the Wait state.



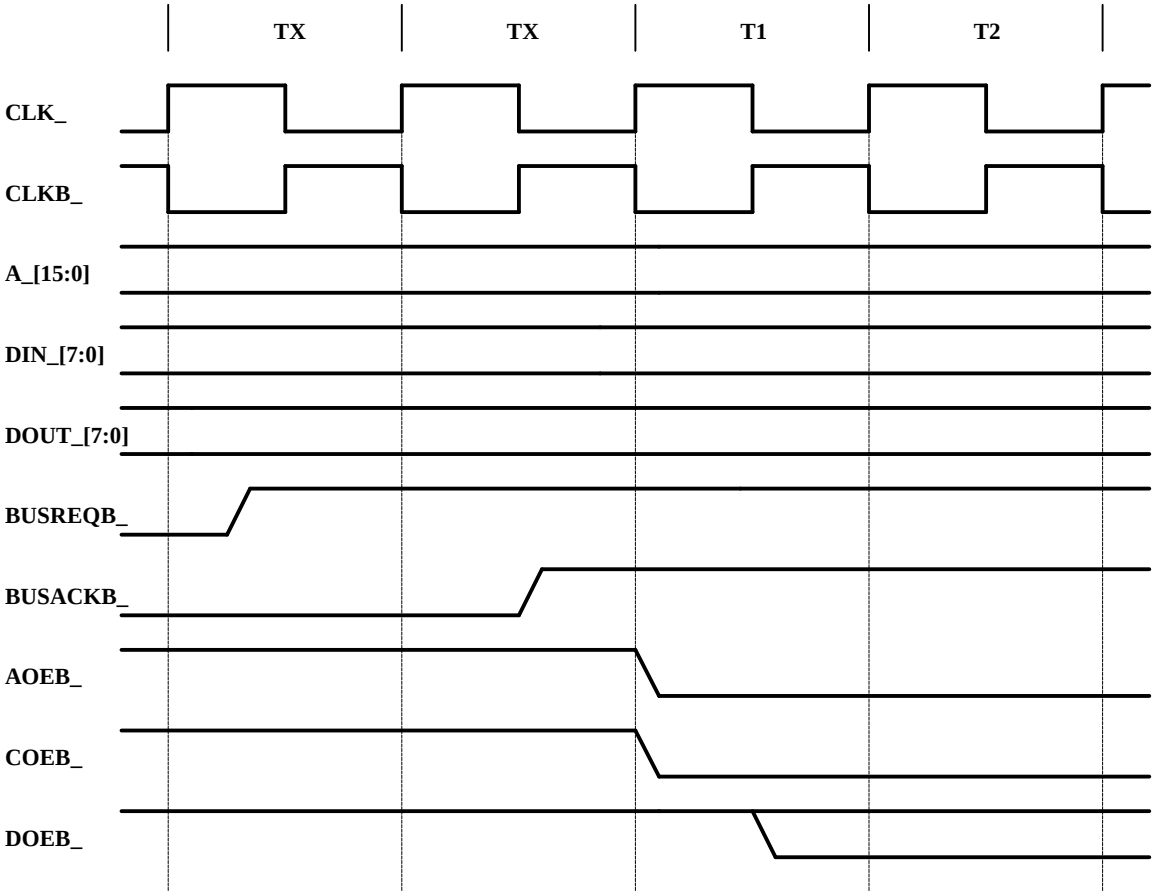
5.7 Bus Request/Acknowledge (Entry)

The timing of the release of processor control of the bus is shown below. The Y180 can release the bus after completion of any machine cycle. None of the Y180 signals actually go floating; rather, the various output enable signals go inactive and the BUSACKB_ signal is activated.



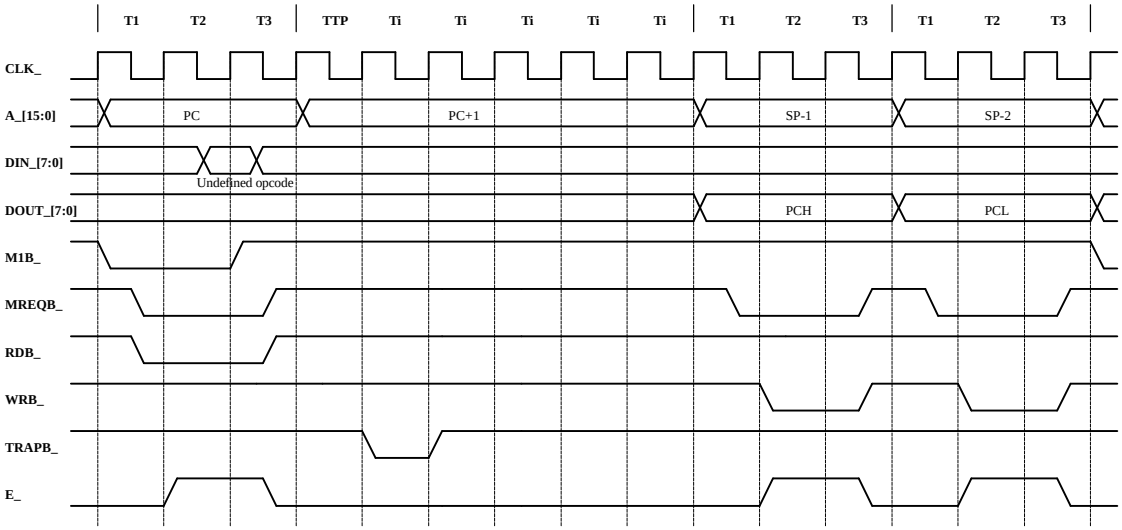
5.8 Bus Request/Acknowledge (Exit)

The timing for resumption of processor control of the bus is shown below. The Y180 can reacquire the bus during any clock cycle of the bus release phase. The various output enable signals go active and the BUSACKB_ signal is deactivated.



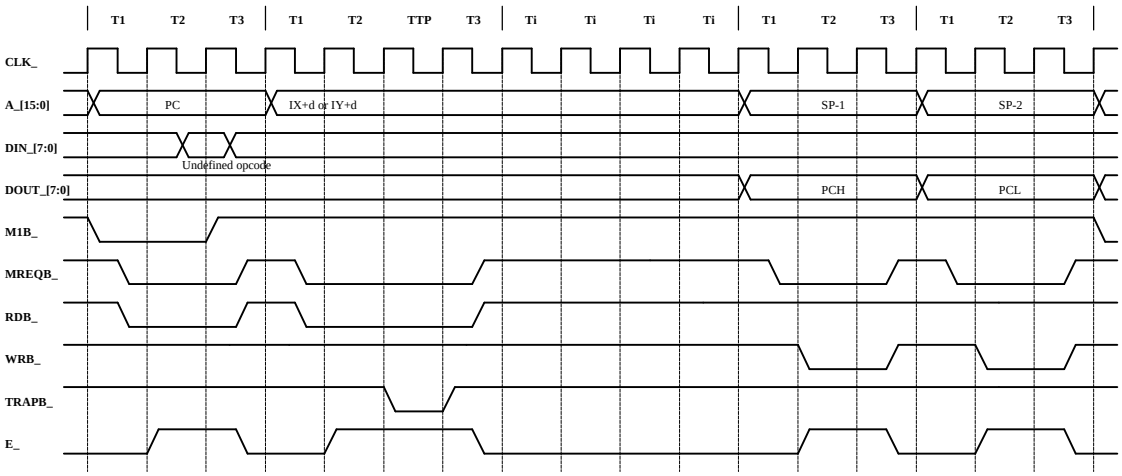
5.9 Trap (second opcode byte)

The timing of an undefined second byte opcode trap is shown below. The fetch of the undefined opcode is followed by the Trap cycle, five internal operation cycles, and two normal write cycles to push the PC of the undefined opcode to the stack. The processor then jumps to location 0000h and starts fetching instructions. The TRAPB_ information should be latched outside the CPU to distinguish this case from the normal reset case. The second byte opcode trap can be distinguished from the third byte opcode trap by the timing of the TRAPB_ signal. The start of the illegal instruction in this case is the stacked PC value minus one.



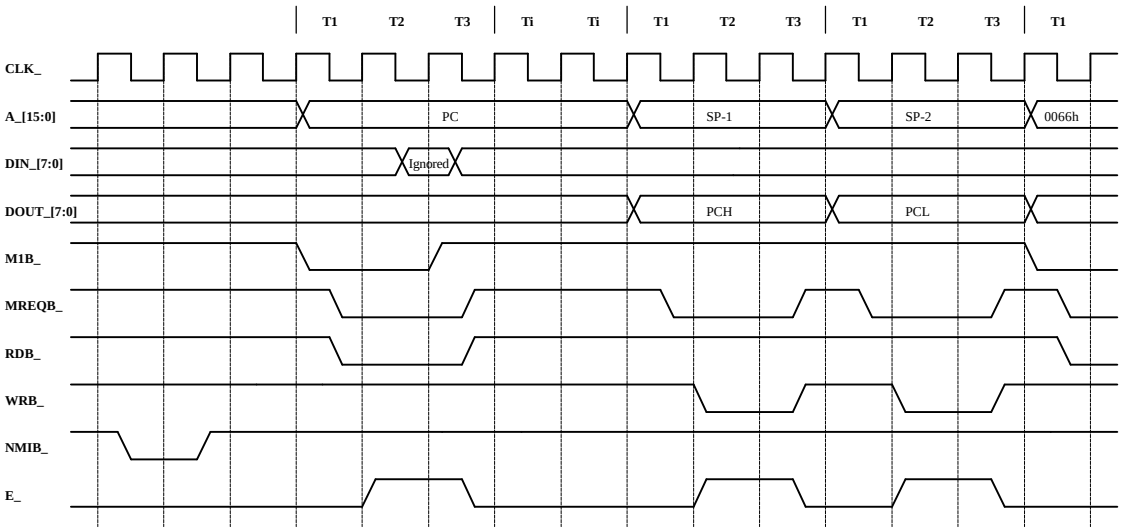
5.10 Trap (third opcode byte)

The timing of an undefined third byte opcode trap is shown below. The fetch of the undefined opcode is followed by the normal Read cycle (all three-byte instructions use indexed addressing) with an embedded Trap cycle, four internal operation cycles, and two normal write cycles to push the PC of the undefined opcode to the stack. The processor then jumps to location 0000h and starts fetching instructions. The TRAPB_ information should be latched outside the CPU to distinguish this case from the normal reset case. The third byte opcode trap can be distinguished from the second byte opcode trap by the timing of the TRAPB_ signal. The start of the illegal instruction in this case is the stacked PC value minus two.



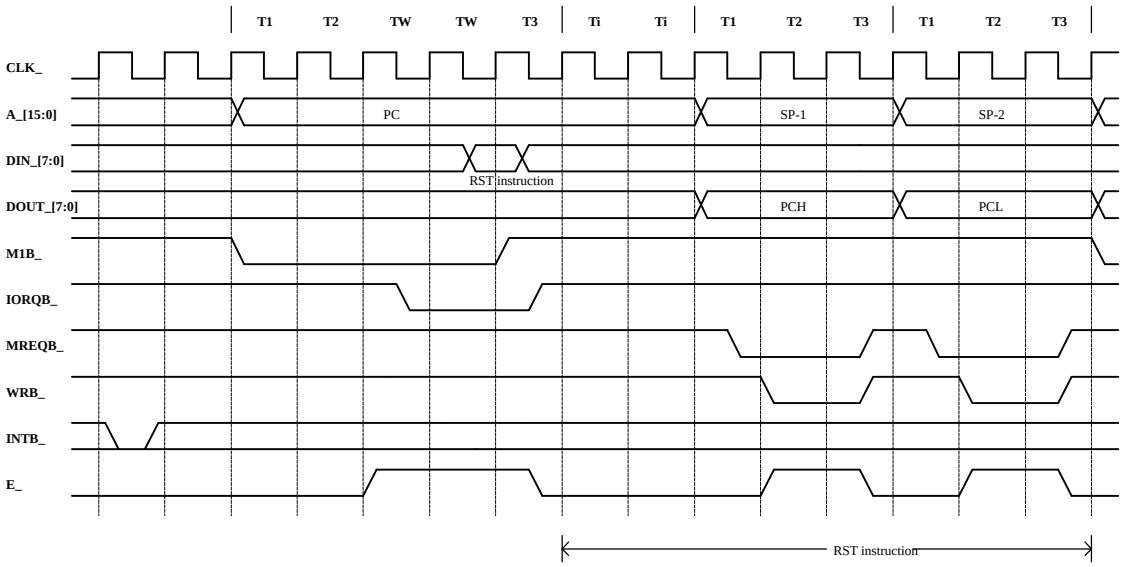
5.11 Non-Maskable Interrupt Acknowledge

The timing of a Non-Maskable interrupt acknowledge cycle is shown below. The NMIB_ input is edge-sensitive and cannot be masked by software. NMIB_ must be sampled Low for two consecutive falling edges of CLK_ to be recognized by the processor. The NMI acknowledge cycle looks exactly like an instruction fetch for the first three clock cycles, except that the data bus is ignored. These three clock cycles are followed by two internal operation cycles and two write cycles to push the contents of the program counter onto the stack. Execution then begins at 0066h with an instruction fetch. The NMI service routine must end with the RETN instruction to properly restore the state of the interrupt enable flag prior to the NMI.



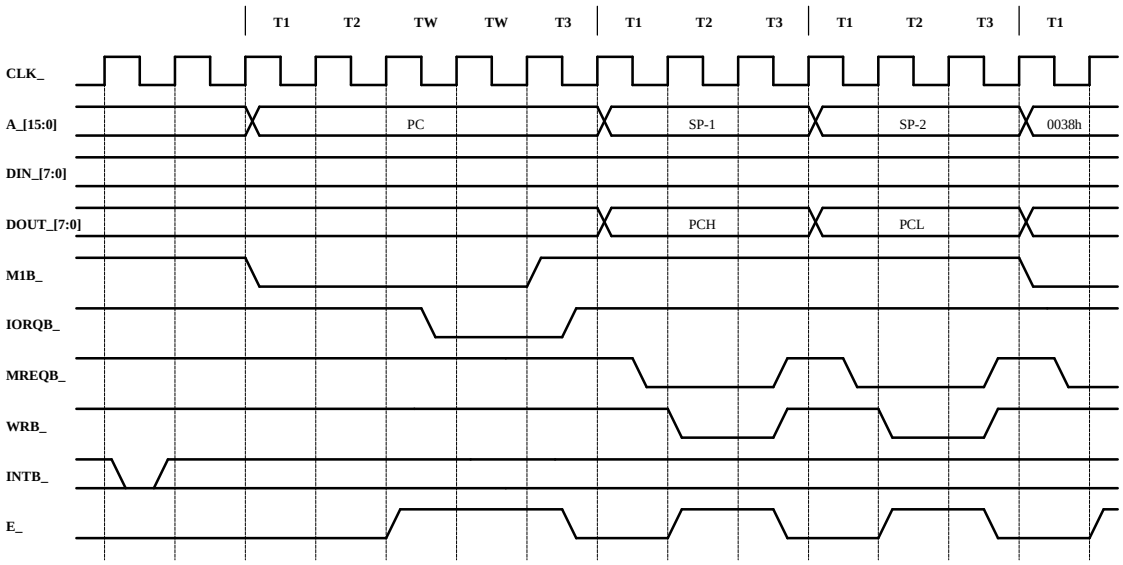
5.12 Mode 0 Interrupt Acknowledge

The timing of a Mode 0 interrupt acknowledge cycle is shown below. The Mode 0 interrupt acknowledge cycle fetches an instruction from the data bus but does not increment the PC, because this instruction is not part of the program. The data bus is sampled by the rising edge of CLK_ at the beginning of T3, just as in a normal instruction fetch cycle. Any instruction can be used, but the most convenient are the Restart (RST) instructions, because they are only one byte and push the PC onto the stack. The figure below shows an RST instruction being fetched during the acknowledge cycle. Note that a Trap is possible if an invalid opcode is fetched during the Mode 0 interrupt acknowledge cycle.



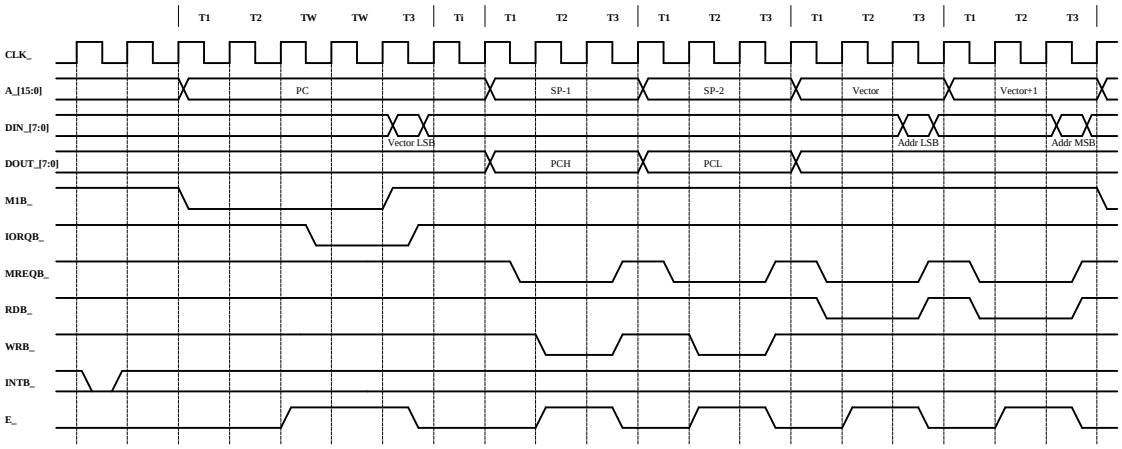
5.13 Mode 1 Interrupt Acknowledge

The timing of a Mode 1 interrupt acknowledge cycle is shown below. The Mode 1 interrupt acknowledge cycle consists of a three clock cycle (plus two automatic Wait states) special bus cycle, followed by two normal write cycles to push the contents of the PC onto the stack. The processor then jumps to location 0038h for the service routine.



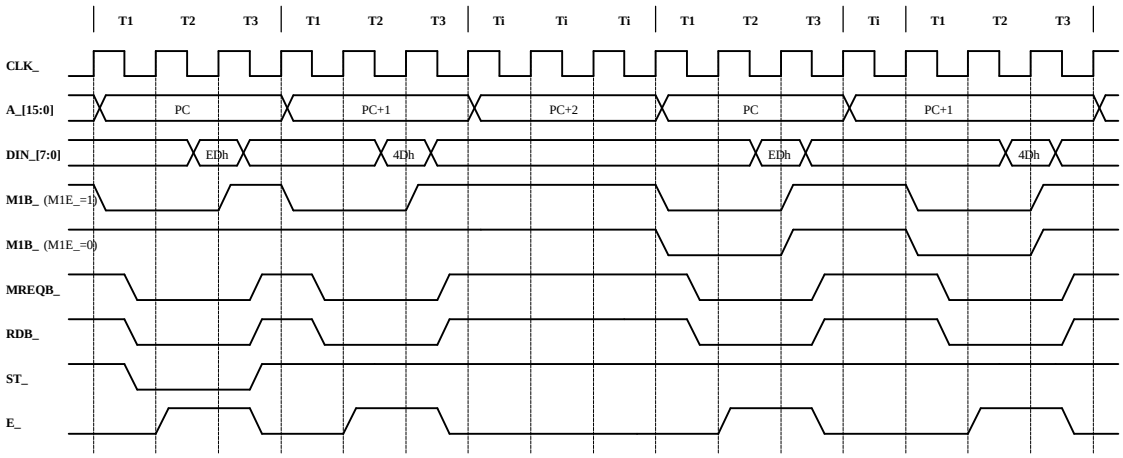
5.14 Mode 2 Interrupt Acknowledge

The timing of a Mode 2 interrupt acknowledge cycle is shown below. The Mode 2 interrupt acknowledge cycle consists of a three clock cycle (plus two automatic Wait states) special bus cycle which reads a vector from the data bus, an internal operation cycle, followed by two normal write cycles to push the contents of the PC onto the stack, followed by two normal read cycles to fetch the interrupt jump table entry corresponding to the vector fetched during the special bus cycle. The processor then jumps to the address fetched from the interrupt jump table for the service routine. The upper eight bits of the interrupt jump table starting address are held in the I register in the processor. Note that the vector must be an even number. That is, the least significant bit of the vector must be a zero.



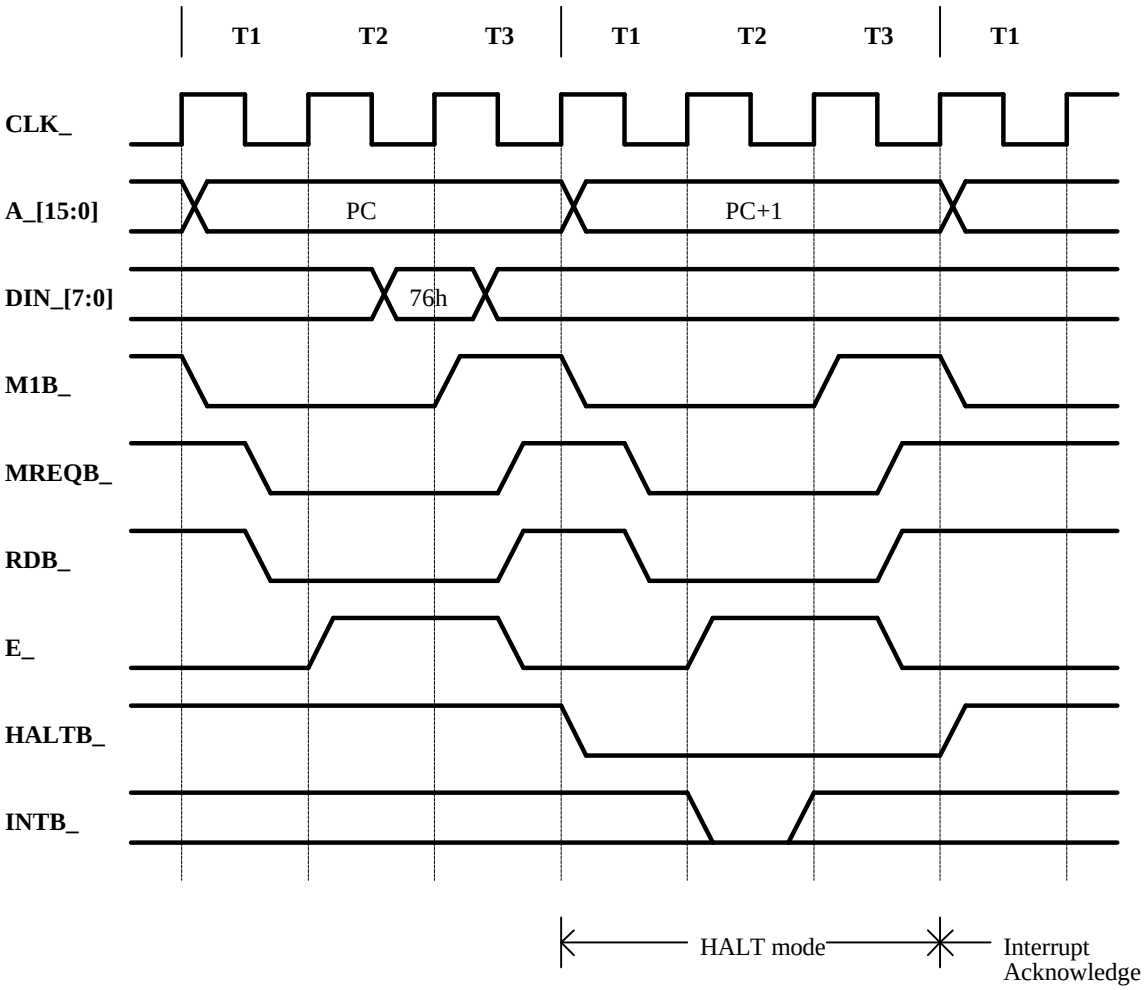
5.15 Return From Interrupt (RETI)

The timing of the RETI instruction sequence is shown below. The Y180 refetches the two byte opcode of the RETI to allow peripheral controllers to recognize the RETI instruction. Proper operation of those peripheral controllers that recognize the RETI instruction requires that the M1E_ input be Low so that only one RETI is recognized.



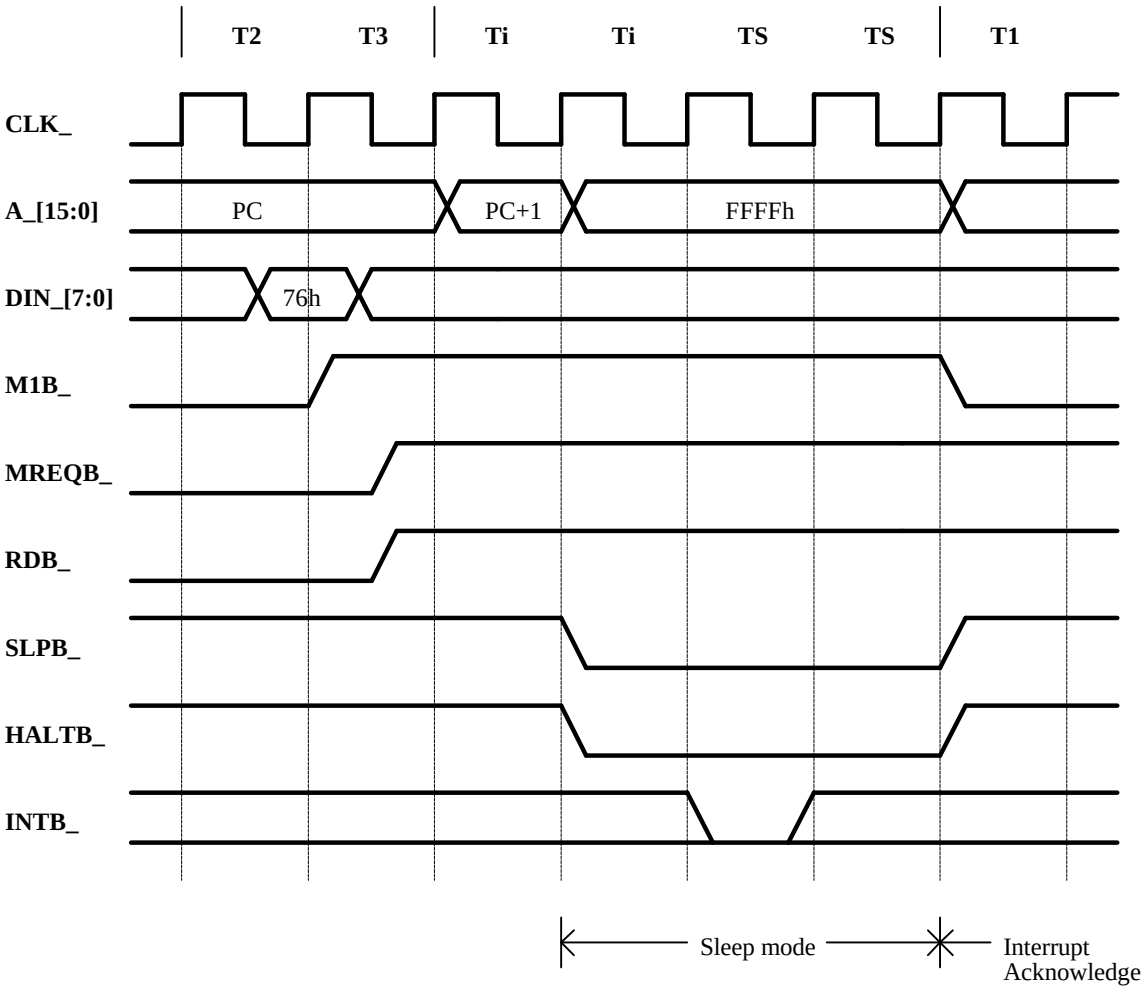
5.16 Halt Entry and Exit

The Halt mode is entered when the HALT instruction is executed, as shown below. In the Halt mode the processor continuously performs Halt cycles, which are three clock cycle bus cycles identical to instruction fetch cycles except that the HALTB_ output is Low. In the Halt mode Bus release (through BUSREQB_ and BUSACKB_) can still occur, but the only way to exit the Halt mode is with either an interrupt (NMIB_ or INTB_) or via reset. The timing for exiting the Halt mode via INTB_ is shown below. Note that INTB_ can only be used to exit the Halt mode interrupts are enabled when the HALT instruction is executed. If the Halt mode is exited via NMIB_ or INTB_, the processor will resume instruction execution (after the interrupt service routine) at the address of the instruction following the HALT instruction.



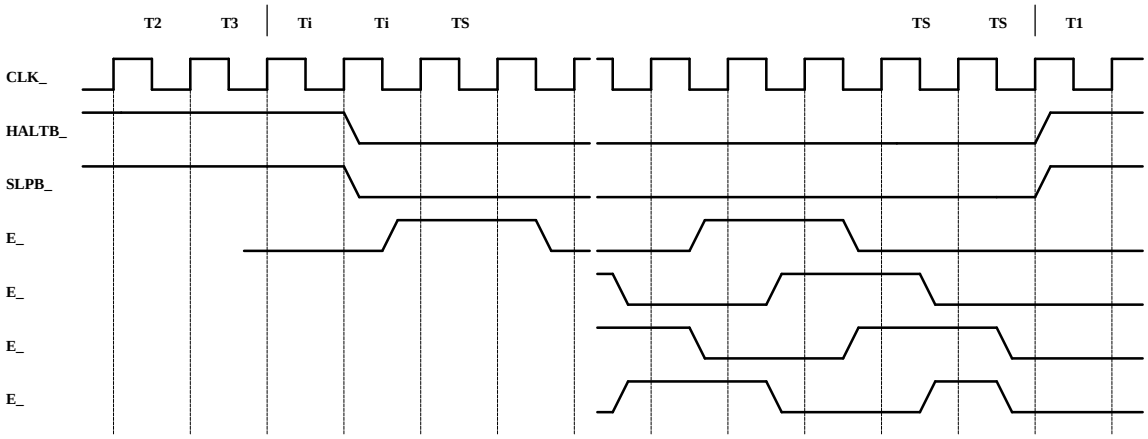
5.17 Sleep Entry and Exit

The Sleep mode is entered when the SLP instruction is executed, as shown below. In the Sleep mode the processor continuously performs Sleep cycles, which are single clock cycle bus cycles identical to internal operation cycles except that both the HALTB_ output and SLPB_ outputs are Low. In the Sleep mode Bus release (through BUSREQB_ and BUSACKB_) can still occur, but the only way to exit the Sleep mode is with either an interrupt (NMIB_ or INTB_) or via reset. The timing for exiting the Sleep mode via INTB_ is shown below. Note that INTB_ can only be used to exit the Sleep mode interrupts are enabled when the SLP instruction is executed. If the Sleep mode is exited via NMIB_ or INTB_, the processor will resume instruction execution (after the interrupt service routine) at the address of the instruction following the SLP instruction.



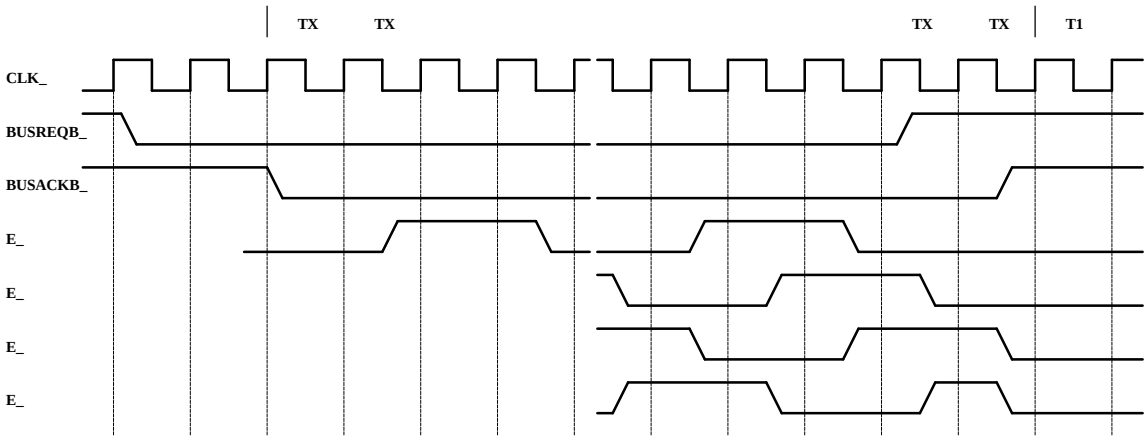
5.18 E_ Signal during Sleep

In the Sleep mode the E_ signal is continuously generated with a four clock-cycle period. The timing of the E_signal is synchronized with the start of the Sleep mode, and terminates cleanly at the end of the Sleep mode, as shown below. This behavior is controlled by a small state machine in the IO_CTRL section of the design and can be eliminated if unnecessary or for further power savings during Sleep mode.



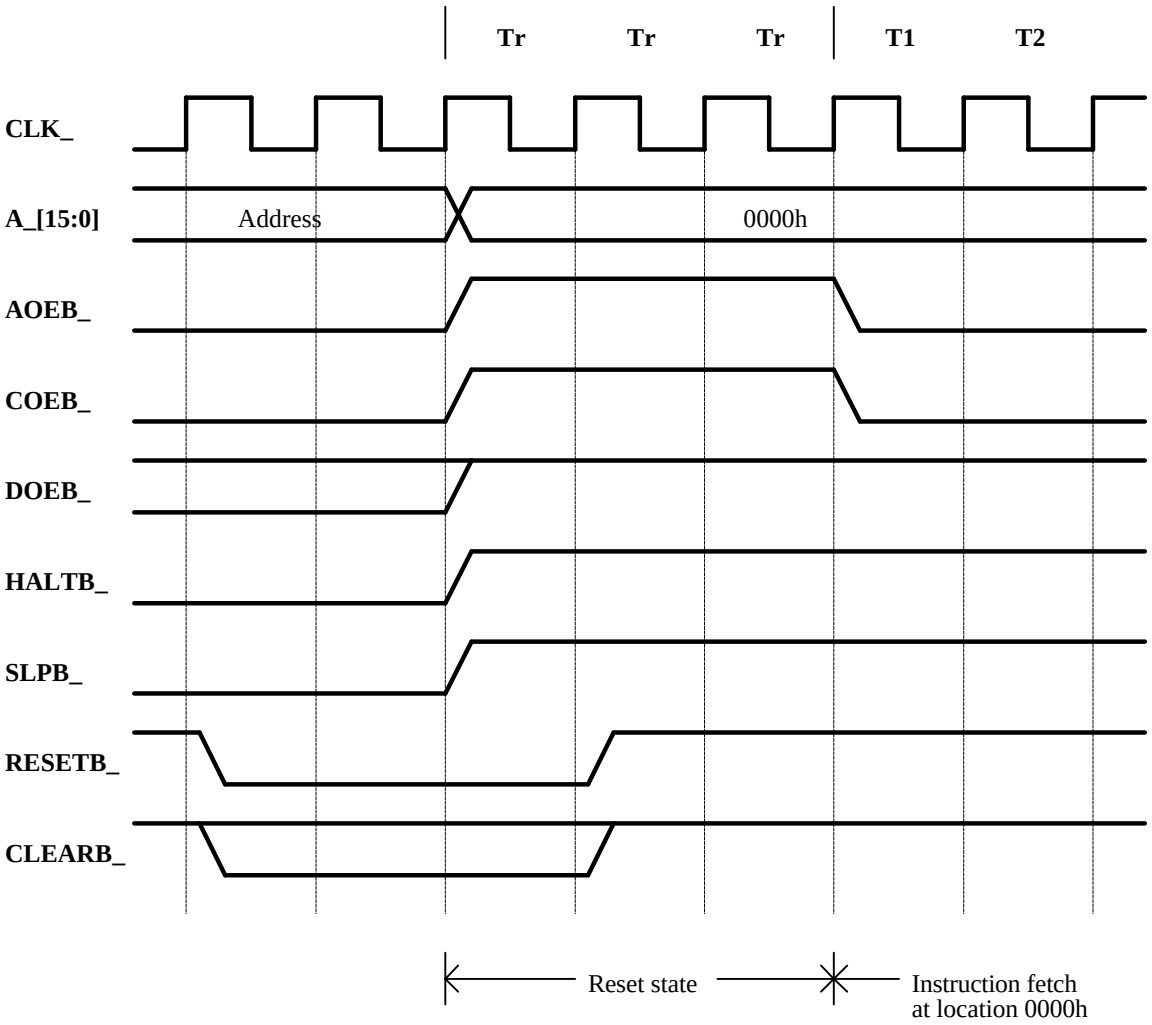
5.19 E_ Signal during Bus Request/Acknowledge

During the release of processor control of the bus the E_ signal is continuously generated with a four clock-cycle period. The timing of the E_signal is synchronized with the start of the bus release, and terminates cleanly when the processor reacquires control of the bus, as shown below. This behavior is controlled by a small state machine in the IO_CTRL section of the design and can be eliminated if unnecessary.



5.20 Reset and Clear

The Reset state is entered when the RESETB_ pin is Low for two consecutive rising edges of CLKB_, as shown below. On the next rising edge of CLK_ after RESETB_ has been sampled Low twice, the Y180 enters the Reset state, independent of the current machine cycle or clock cycle. It remains in the Reset state until after RESETB_ is sampled High. At that time, the Y180 begins fetching instructions from location 0000h. The Reset state clears all of the state machines internal to the Y180. It also resets the PC, SP, I and R registers, selects Interrupt Mode 0, and disables the maskable interrupts. If CLEARB_ is asserted Low coincident with RESETB_, all of the other registers in the Y180 are reset also. RESETB_ should always be asserted on power-up, and may be used to exit from the Halt mode or Sleep mode also.



6 Differences

This section describes the differences between the Y180 and the Z80180. All of the differences are related to input or output timing or operation. All instruction results and clock cycle timing are identical between the two devices.

AOENB_ The Y180 provides an AOENB_ output for control of an external 3-state Address bus. The Y180 Address Bus, A_[15:0], is always driven. The Z80180 address bus is 3-state.

CLEARB_ The Y180 provides a CLEARB_ input to initialize all of the register file. This input can be quite useful for simulation, but is not strictly necessary for the final design and can be tied High with no ill effect. The Z80180 has no such reset mechanism.

CLK_ The Y180 requires both CLK_ and CLKB_, although the CLKB_ input is used only in the I/O interface module. CLKB_ is required to match the timing characteristics of the Z80180, which changes outputs and samples inputs on both edges of the clock.

COENB_ The Y180 provides a COENB_ output for control of external 3-state buffers on the control signals. Y180 control signals are always driven. The Z80180 control signals are 3-state.

DOENB_ The Y180 employs separate data input and output busses, DIN_[7:0] and DOUT_[7:0], and a Data Output Enable signal, DOENB_, to control an external bidirectional bus, if desired. The DOUT_ bus changes only on the rising edge of CLK_. The Z80180 employs a bidirectional data bus, and in the output mode the leading edge of the data changes on the falling edge of the clock. The timing of the Y180's DOENB_ signal is such that the timing of an external bidirectional bus, if implemented, will match that of the Z80180.

IOCB_ The Y180 utilizes an IOCB_ input to control the timing of the RDB_ and IORQB_ outputs during I/O operations. In the Z80180, the timing of these two signals is controlled by a bit in a register external to the CPU. This register can be created for complete compatibility, and its output tied to the IOCB_ input of the Y180.

M1E_ The Y180 utilizes an M1E_ input to control the operation of the M1B_ signal for compatibility with the Z80180. In the Z80180, the operation of the /M1 signal is controlled by a bit in a register external to the CPU. This register can be created for complete compatibility, and its output tied to the M1E_ input of the Y180.

NMIB_ The Y180 requires that NMIB_ be Low for two consecutive rising edges of CLKB_ after being sampled High. The latest that these two rising edges of CLKB_ can occur, and still be accepted at the end of the current machine cycle, is during the two clocks preceding the last clock cycle of a machine cycle. This different than the timing for the Z80180, which catches the falling edge of /NMI as late as one-half clock before the last clock cycle of a machine cycle. The Y180 timing is more robust, acting as "glitch filter" on this edge-sensitive input.

RESETB_ The Y180 requires that RESETB_ be Low for two consecutive rising edges of CLKB_, and responds on the next rising edge of CLK_ after it is sampled Low for the second time. On exiting the Reset state, the Y180 starts fetching the instruction at location 0000h one and one-half clock cycles after sampling RESETB_ High. This is different from the six clock cycle minimum Low time requirement and the two and one-half clock cycle response time for the Z80180.

SLPB_ The Y180 provides a separate SLPB_ output to indicate that the device is in the Sleep mode. The Z80180 requires decoding of the state of several outputs to indicate that the device is in Sleep mode. Note that the Y180 provides the same encoding on the outputs, but using the SLPB_ output is easier.

TRAPB_ The Y180 provides a separate TRAPB_ output to indicate that the device has fetched an illegal opcode. The two different cases of a Trap can be distinguished by the state of the RDB_ signal when TRAPB_ is Low. The Z80180 provides a register, outside of the CPU to hold the trap information. This external register can be easily created and written via the TRAPB_ signal for compatibility.

7 Future Enhancements

The basic Z80 architecture has plenty of room for expansion, having been created when transistor budgets were extremely limited. The Y180, as a Verilog HDL model, is quite simple to upgrade or change. Some of the possible variants of the Y180 are described below.

One possible performance enhancement would be in the area of the ALU. If it were widened to 16 bits, for example, the two clock cycle penalty for calculating the indexed address would be reduced to one clock cycle. This would also allow the instruction set to be expanded to provide a full complement of 16-bit instructions. Similarly, dedicated multiply logic can be added to reduce the time required for the multiply instruction. It currently uses a shift-and-add technique.

Another straightforward enhancement is an expansion of the address space beyond the current 64K byte limit. This requires widening at least some of the registers and modifying such instructions as PUSH, POP, CALL and RETURN, but this is a much cleaner solution than adding an MMU like the Z80180 does. This is the first extension planned for the Y180. Note that an MMU (even your own design) can still be easily added to the basic Y180.

All of the registers in the Y180 can be widened to a full 32-bits. In addition, it is a simple modification to add more banks of registers, or dedicated registers that interface to specialized logic to perform some application-specific function such as a text search, encryption/decryption, independent I/O transfer, and so on.

Of course, it is a simple matter to modify the interface of the Y180 to fit specific requirements of timing, signal polarity, and so on. Even the automatic wait-state inserted for I/O access requires only a single line of Verilog HDL code to be changed.

The Y180 is just the first in a series of designs. Specific future versions are already planned, but if you have a requirement, please let us know.

8 Model Organization

The organization of the Y180 Verilog HDL model is identical to that shown in the block diagram of section 3.1. That is, there is a Top Level Module which contains the four main modules of the device. Each module is flat, except for the Address and Data Module, which uses two byte-wide register modules. Even though there are only three hierarchical levels in the overall model, each module is structured into a number of self-contained sections for easy modification.

Symbolic label definitions are used, rather than hard encoding, in almost all cases in the design. Those cases where the hard encoding is used are listed below. In all cases where hard encoding is not used, the symbolic label definitions can be changed, to provide unencoded signals, or just different encodings. When modifying symbolic label definitions, only the `include file that contains all of the parameter definitions needs to be modified.

8.1 Y180_TOP (Top Level Module)

Y180_TOP is the Top Level Module for the device. It contains only the pins and the four main modules of the Y180. Note that no symbolic labels are used at this level, and all of the pins of the device use capital letters followed by an underscore.

8.2 PARAMS (Parameter Definition `include File)

PARAMS is the parameter definition `include file for the device. It contains all of the symbolic label definitions used in the design and is called with an `include in each of the four main modules of the device. If you want to modify the symbolic label definitions, only this file needs to be modified. As mentioned previously, some of the encodings must not be modified. These are described below, and are clearly marked with warning comments in this file.

The page register encoding, which identifies which code page the instruction is on, must not be modified, as it seldom treated symbolically. The encoding for the page register has been carefully chosen to simplify the decoding of groups of similar instructions on different pages.

The clock cycle encoding, which identifies the particular clock cycle within a machine cycle, must not be modified. This encoding was chosen very specifically to minimize the number of bits which change where the next state is conditional on some signal.

The register address encoding should not be modified, unless the encoded register address generators in the Central Control Module are also modified, because a portion of the address is taken directly from bits in the opcode in these address generators.

Obviously, the definition of TRUE and FALSE should not be changed.

8.3 IO_CTRL (I/O Interface Module)

IO_CTRL is the I/O Interface Module for the device. This module translates between the external pins and the internal busses and signals of the Y180. This is the only module which uses CLKB_ and consists primarily of flip-flops to translate timing. This is where RESETB_ is translated into the internal signal resetb. Because resetb goes to nearly every flip-flop in the Y180, it will be heavily loaded. This is the only signal, besides the clock, that will require special attention when implemented.

8.4 M_STATE (Machine State Module)

M_STATE is the Machine State Module for the device. This module contains the machine cycle state machine, the clock cycle state machine, and the interrupt enable and mode flip-flops. As mentioned previously, the clock cycle state machine was carefully designed to minimize state transitions and should not be modified.

8.5 CTR_CTL (Central Control Module)

CTR_CTL is the Central Control Module for the device. This module is purely combinatorial, and can be implemented as either random logic, microcode, or a combination of both. The only inputs to this module are the page register, the instruction register, the machine cycle state and the clock cycle state.

8.6 DATA_IO (Address and Data Module)

DATA_IO is the Address and Data Module for the device. This module contains the ALU, Program Counter, Instruction Register, Page Register, Flag Registers, register file and temporary registers. This is where all of the address and data manipulation is done in the Y180. This is the only module in the Y180, other than the Top Level Module, which contains other modules.

8.7 REG_BYTE (Byte-wide Register in the Register File)

REG_BYTE is a byte-wide register for use in the register file. A unique register is used for the register file to allow it to be replaced with something other than flip-flops if desired. Note that the majority of registers in the register file are reset by the clearb signal, which is derived from the CLEARB_ input. If CLEARB_ is not used in your design, these registers do not need to allow for reset. The Stack Pointer, I and R registers are reset by the resetb signal, however.

8.8 REG_8BIT (Byte-wide General-Purpose Register)

REG_8BIT is a byte-wide general-purpose register for use other than in the register file. It is merely a grouping of eight flip-flops that is used for convenience in the Verilog HDL description.

9 Test Suite

The Y180 Verilog HDL model includes a complete test suite to verify proper operation of the device both before and after implementation. The test suite verifies the proper operation of every valid instruction, trap on every illegal opcode, proper operation with and without Wait states for every instruction, proper operation with Bus Request before and after every possible machine cycle, all interrupt modes and proper flag operation. Running the test suite requires virtually no user intervention.

The test suite does not test every instruction in conjunction with interrupt and NMI, but rather checks every group of instructions sharing a common interrupt or NMI Verilog description. The test suite does not currently check for the proper timing of every input and output. This was done manually during the development of the test suite, and the model is believed correct as supplied. If exhaustive input and output timing verification is desired, the top level model can be modified to check this.

It is a relatively straightforward process to trace the inputs and output during simulation to generate vector files suitable for use with ATE testers. Another alternative is to allow the synthesizer to insert scan test logic during the synthesis process.

9.1 TOP_LEV (Top Level for Simulation)

TOP_LEV is the top level module for simulation. It contains the Y180 module itself, a read memory which is loaded with the program to be executed, a compare memory which is loaded with the compare data for the program, the clock generator, a pair of reset tasks, a couple of tasks useful for debugging, interrupt and NMI generators, a Wait generator, a Bus Request generator, and a compare error flag and counter.

The top level as supplied runs through the entire test suite without Wait or Bus Request, followed by a pass with one wait state in every bus cycle, followed by a pass where Bus Request is active all the time and is released for one clock cycle at a time to allow only one machine cycle to be executed between bus requests. The Bus Request pass is several times longer than an individual pass and can be eliminated if necessary by editing the file so that the patterns are not executed while the variable DISABLE_BREQ is zero.

9.2 SETUP_HL (Initialization Pattern)

SETUP_HL is a short pattern used to initialize the HL register pair before starting the first pattern. Executing this pattern first makes it possible to rearrange the order of the remaining patterns. This is because several of the patterns require HL to contain a jump address at the start of the pattern. In a similar fashion, the HL register pair is initialized at the end of every pattern. Every pattern ends with what would be an infinite loop at location 0C0h. This loop is detected by a test in TOP_LEV and used to load the next pattern. Any patterns that you add to the test suite should attempt to follow this convention.

9.3 INT_OPS (Interrupt Operation)

INT_OPS checks all of the interrupt modes and NMI for all of the possible cases. This pattern is also used to check that the two input options, M1 Enable and I/O Control are functioning correctly. Sleep mode and Halt mode are also checked in this pattern.

9.4 ALU_OPS (ALU Operation)

ALU_OPS checks all of the data manipulation instructions and flag results. Every data manipulation instruction is individually checked, usually more than once, to ensure both proper operation and flag results. Both byte and 16-bit instructions are checked in this pattern.

9.5 DAT_MOV (Data Movement Operation)

DAT_MOV checks all of the data movement instructions, both internal and external. Every data movement instruction is individually checked, usually more than once, to ensure both proper operation and no adverse consequences (improper decoding, for example). Both byte and 16-bit instructions are checked in this pattern, but the block move instructions are checked in a separate pattern.

9.6 TRP_2ND (Trap on Second Byte Operation)

TRP_2ND checks all of the two-byte illegal opcodes. Each two-byte illegal opcode is individually checked for a trap and no adverse consequences.

9.7 TRP_3RD (Trap on Third Byte Operation)

TRP_3RD checks all of the three-byte illegal opcodes. Each three-byte illegal opcode is individually checked for a trap and no adverse consequences.

9.8 BIT_OPS (Bit Manipulation Operation)

BIT_OPS checks all of the bit operations. Each bit operation instruction is individually checked for both proper operation and proper flag results, with no adverse consequences.

9.9 JMP_OPS (Jump Operation)

JMP_OPS checks all of the program flow instructions. Each Jump or Call instruction is individually checked, including the taken/not taken case if it is a conditional instruction. This is where Restart, Return and DJNZ are checked.

9.10 IO_OPS (I/O Operation)

IO_OPS checks all of the individual and block I/O instructions. The block move instructions are also checked here. Both the looping and terminating case of the block instructions are checked.

10 Installation

The Y180 Verilog HDL Model was developed on a PC, using Microsoft Works for Windows 2.0 for the spreadsheet, Microsoft Publisher for the text, and Veriwell for Windows 2.0 for the verification. The model uses only synthesizeable constructs and contains nothing unique to the simulator used for the development.

The standard method of providing the Y180 Verilog HDL Model is as text files on a single 3.5" HD disk, since this will be compatible with the majority of destinations. The file structure of this disk is shown below.

The design spreadsheet will be provided in Microsoft Works for Windows 2.0 format files (zipped) on a 3.5” HD disk. Should the documentation (this manual) be required in machine-readable format other than the default Acrobat Portable Document Format (pdf), it will be provided in Microsoft Publisher format (zipped) on a 3.5” HD disk.

10.1 File structure

```
design  .....ctr_ctl.v
        .....data_io.v
        .....io_ctrl.v
        .....m_state.v
        .....params.v
        .....reg_8bit.v
        .....reg_byte.v
        .....top_lev.v
        .....y180_top.v
memory  .....alu_ops.vm
        .....alu_opsd.vm
        .....bit_ops.vm
        .....bit_opsd.vm
        .....dat_mov.vm
        .....dat_movd.vm
        .....int_ops.vm
        .....int_opsd.vm
        .....io_ops.vm
        .....io_opsd.vm
        .....jmp_ops.vm
        .....jmp_opsd.vm
        .....setup_hl.vm
        .....trp_2nd.vm
        .....trp_2ndd.vm
        .....trp_3rd.vm
        .....trp_3rdd.vm
testing .....alu_ops.s
        .....alu_opsd.s
        .....bit_ops.s
        .....bit_opsd.s
        .....dat_mov.s
        .....dat_movd.s
        .....int_ops.s
        .....int_opsd.s
        .....io_ops.s
        .....io_opsd.s
        .....jmp_ops.s
        .....jmp_opsd.s
        .....setup_hl.s
        .....trp_2nd.s
        .....trp_2ndd.s
        .....trp_3rd.s
        .....trp_3rdd.s
```