

Y90-180 Microprocessor

Technical Manual

Systemyde International Corporation



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.

Every effort has been made to ensure the accuracy of the information contain herein. If you find errors or inconsistencies please bring them to our attention. In all cases, however, the Verilog HDL source code for the Y90 design defines “proper operation”.

Copyright © 2010, Systemyde International Corporation. All rights reserved.

Notice:

“Z80”, “Z180” and “Zilog” are registered trademarks of Zilog, Inc. All uses of these terms in this document are to be construed as adjectives, whether or not the noun “microprocessor”, “CPU” or “device” are actually present.

Table of Contents

Introduction	3
Features	5
Pin Descriptions	7
External Timing	13
Instruction Set	33
Memory Management Unit (MMU)	135
Interrupt Control	139
Direct Memory Access (DMA)	141
Asynchronous Serial Communications Interface (ASCI)	149
Programmable Reload Timer (PRT)	167
Clocked Serial I/O (CSIO)	161
System Functions	165
I/O Register Addresses	169
Top level Verilog Code	173

Revision History

Date	Changes	Page(s)

Introduction

This book documents the operation of the Y90-180 microprocessor. The Y90-180 design is supplied in Verilog HDL and can be implemented in any technology supported by a logic synthesis tool that accepts Verilog HDL. Included in the design package is a test bench that exercises all instructions, flag settings, and representative data patterns. The test patterns should achieve at least 95% fault coverage.

The Y90-180 was designed in a clean-room environment and is an upgrade of the Zilog Z180 series of microprocessors. Only publicly available documentation was used to create this design so there may be minor differences where the public documentation is misleading or lacking. The instruction execution times are not identical between the two designs. The Y90-180 operates with a consistent two-clock-cycle machine cycle, while the Z180 microprocessors use machine cycles that vary from three to seven clock cycles.

The Y90-180 design, depending on the version, may not implement all of the features or operating modes of the Z180 architecture. In particular, almost all of the added peripheral features present in the Z8S180 and Z8L180 are also present in the Y90-180. The Y90-180 contains the peripheral functions present in the original Z80180.

This document should always be used as the final word on the operation of the Y90-180, but it is useful to refer to the Zilog documentation if the description given here is too cryptic. The Z180 architecture is over thirty years old, so it is assumed that it is already at least somewhat familiar to the reader.

The Y90-180 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 knowledgeable users the opportunity to customize the design for unique application requirements.

Features

- * Fully functional synthesizable Verilog HDL version of the Z180 MPU
- * Vendor and technology independent
- * 189 instructions, with eight addressing modes
- * 64K byte (1M byte with MMU) memory addressing capability
- * Separate 64K byte I/O address space
- * 16 bit ALU with bit, byte and BCD operations
- * Powerful vectored interrupt capability with separate interrupt vector input bus
- * Static, fully synchronous design uses no 3-state buses
- * Uniform 2 clock-cycle machine cycle
- * Includes peripheral functions present in the original Z180:
 - Two Direct Memory Access (DMA) channels
 - Two Asynchronous Serial Communications Interface (ASCI) channels
 - Two Programmable Reload Timer (PRT) channels
 - One Clocked Serial I/O (CSIO) channel
- * Memory interface matches common FPGA and ASIC memory timing
- * Separate I/O bus, compatible with AMBA Peripheral Bus
- * Illegal instruction detection
- * Full design documentation included
- * Verilog simulation and test suite included

Shown below are the registers 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 Y90-180 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.

A	F
B	C
D	E
H	L
IX	
IY	

Main Register Bank

A'	F'
B'	C'
D'	E'
H'	L'

Alternate Register Bank

I	R
----------	----------

Special Function Registers

SP
PC

Pin Descriptions

The Y90-180 design does not attempt to match the signals or timing present on the Z180 microprocessor. Rather, the interfaces and signals are optimized for use in either an ASIC or an FPGA.

Memory and I/O use separate address and data buses in addition to the separate control signals. The memory bus is designed to match typical ASIC and FPGA memory timing, although it can be used with stand-alone memory devices just as easily. A separate interrupt vector bus is provided for use with an interrupt controller. If desired, this interrupt vector bus can be tied to either the memory or I/O input bus for operation more closely resembling that of the original Z180.

The original Z180 microprocessor multiplexes a number of pin functions because of package constraints. The Y90-180 makes no attempt to duplicate this signal multiplexing.

The interface signals for the Y90-180 are detailed below. Note that all inputs except the two resets are sampled by the rising edge of the clock and all outputs change in response to the rising edge of the clock.

cka0_in, cka1_in (inputs, active-High) The Async Clock In signals are the external clock inputs for the two ASCI channels.

cka0_out, cka1_out (outputs, active-High) The Async Clock Out signals are the internal clock outputs for the two ASCI channels.

cks_in (input, active-High) The Clocked Serial Clock In signal is the external clock input for the CSIO channel.

cks_out (output, active-High) The Clocked Serial Clock Out signal is the internal clock output for the CSIO channel.

clearb (input, active-Low) The Master (test) Reset signal is used to initialize all of the flip-flops that are not initialized by the user reset signal. Most user-visible registers are not affected by the user reset, so this signal allows full initialization for testing and simulation. This is an asynchronous signal that should be used for Power-On Reset.

- clk** (input, active-High) The CPU Clock connects to all flip-flops in the design.
- cts0b, cts1b** (inputs, active-Low) The Clear To Send signals are the transmit enable modem controls for the two ASCII channels.
- dcd0b, dcd1b** (inputs, active-Low) The Data Carrier Detect signals are the receive enable modem controls for the two ASCII channels.
- dma_ack** (output, active-High) The DMA Acknowledge signal is activated to indicate that the processor has halted to allow another bus master to use the bus. The **iack_tran, io_addr_out, io_data_out, io_tran, mem_addr_out, mem_data_out, mem_tran, reti_tran** and **t1** signals are all inactive (Low) during this time, but the internal DMA controller will drive these signals to perform data transfers. The processor will signal **dma_ack** while in the Halt or Sleep state without de-asserting the **halt_tran** or **sleep_tran** signals. Interrupts are not sampled while the **dma_ack** signal is active, so the exit from a coincident Halt or Sleep state will be deferred until the **dma_ack** signal is no longer active.
- dma_req** (input, active-High) The DMA Request signal requests that the processor halt to allow an external bus master to transfer data on the bus. The processor only releases the bus between instructions, rather than between individual bus transactions. An external bus master has higher priority than the internal DMA controller.
- dreq0b, dreq1b** (inputs, active-Low) These DMA Request signals are used by the internal DMA controller to trigger or gate DMA transfers. These signals may be programmed to be edge- or level-sensitive.
- en_prftch** (input, active-High). The Enable Prefetch signal enables the prefetch operation. Although the prefetch mode can be changed dynamically, it is recommended that this signal be tied either High or Low. The prefetch mechanism increases performance by prefetching an opcode byte during any address calculation time. Only the prefix byte (0xCB, 0xDD, 0xED or 0xFD) of a multi-byte instruction can actually be used after being prefetched.
- halt_tran** (output, active-High) The Halt Transaction signal is activated by the Halt instruction. While in the Halt state the CPU freezes and waits for an interrupt. The **iack_tran, io_addr_out, io_data_out, io_tran, mem_addr_out, mem_data_out, mem_tran, reti_tran** and **t1** signals are all inactive (Low) during this time.
- iack_tran** (output, active-High) The Interrupt Acknowledge Transaction signal is activated to identify an interrupt acknowledge bus transaction. An interrupt

acknowledge occurs in response to either a Non-Maskable Interrupt request or an enabled Maskable Interrupt request. During an interrupt acknowledge the interrupt vector data bus is sampled, although the sampled value is only used in Interrupt Mode 0 or 2 with a maskable interrupt request.

int0_req (input, active-High) The Interrupt 0 Request signal is the highest-priority maskable interrupt request. Maskable interrupts can be enabled and disabled under program control. This interrupt request is not latched, so it should remain active until an interrupt acknowledge transaction occurs. This interrupt request uses an externally-supplied interrupt vector in Interrupt Mode 0 or 2.

int1_req, int2_req (inputs, active-High) The Interrupt 1 (and 2) Request signals are additional maskable interrupt requests. These two interrupt requests use an internally-generated interrupt vector and force Interrupt Mode 2.

io_addr_out (output, 16-bit bus) The I/O Address Output bus carries the address of the I/O port during an I/O transaction. This bus holds the current value until the next I/O transaction or until the **dma_ack** signal is activated.

io_data_in (input, 8-bit bus) The I/O Data Input bus is sampled during the various I/O input instructions. A separate bus allows peripherals to be connected without loading the memory data bus.

io_data_out (output, 8-bit bus) The I/O Data Output bus carries the output data for I/O output instructions. This bus holds the current value until the next output instruction or until the **dma_ack** signal is activated.

io_read (output, active-High) The I/O Read signal indicates the direction of data transfer during I/O transactions. High signals read and Low signals write. This signal is valid only during I/O transactions.

io_strobe (output, active-High) The I/O Strobe signal is one clock cycle wide (in the absence of Wait states) and identifies the data transfer clock cycle for I/O transactions.

io_tran (output, active-High) The I/O Transaction signal is activated for all I/O transactions.

ivec_data_in (input, 8-bit bus) The Interrupt Vector Data Input bus is sampled during interrupt acknowledge transactions. If the interrupt acknowledge was for a maskable interrupt and the CPU is in Interrupt Mode 2, this vector is used as a pointer in the interrupt vector table to find the starting address of the interrupt service routine. In Interrupt Mode 0 the vector is a one-byte RST instruction.

ivec_read (output, active-High) The Interrupt Vector Read signal is one clock cycle wide (in the absence of Wait states) and identifies the data transfer clock cycle for interrupt acknowledge transactions.

mem_addr_out (output, 16-bit bus) The Memory Address Output bus carries the address during memory read and write transactions. This bus holds the current value until the next memory transaction.

mem_data_in (input, 8-bit bus) The Memory Data Input bus is sampled during memory read transactions. A separate bus allows peripherals to be connected without loading the memory data bus.

mem_data_out (output, 8-bit bus) The Memory Data Output bus carries the output data for memory write transactions. This bus holds the current value until the next output instruction.

mem_rd (output, active-High) The Memory Read signal is one clock cycle wide (in the absence of Wait states) and identifies the data transfer clock cycle for memory read transactions.

mem_tran (output, active-High) The Memory Transaction signal is activated for memory read and write transactions. The **mem_tran** signal is active during the Halt state but is inactive during the Sleep state and during DMA transfers.

mem_wr (output, active-High) The Memory Write signal is one clock cycle wide (in the absence of Wait states) and identifies the data transfer clock cycle for memory write transactions.

nmi_req (input, active-High) The Non-Maskable Interrupt Request signal unconditionally interrupts the CPU. This request is internally latched, so that it can be as short as one clock cycle wide.

resetb (input, active-Low) The User Reset signal is used to initialize all state flip-flops and some user registers (the I, R, PC and SP registers). This is an asynchronous signal.

reti_tran (output, active-High) The Return From Interrupt transaction signal is activated immediately after the second stack read transaction during the Return From Interrupt (RETI) instruction. This signal may be used by an external interrupt controller to re-enable interrupts, for example.

rts0b, rts1b (outputs, active-Low) The Request To Send signals are modem control outputs from the the two ASCII channels.

rx_{a0}, **rx_{a1}** (inputs, active-High) The Asynchronous Receive Data signals are the serial data inputs for the two ASCII channels.

rxs (inputs, active-High) The Synchronous Receive Data signal is the serial data input for the CSIO channel.

sleep_tran (output, active-High) The Sleep Transaction signal is activated by the Sleep instruction. While in the Sleep state the CPU freezes and waits for an interrupt. The **iack_tran**, **io_addr_out**, **io_data_out**, **io_tran**, **mem_addr_out**, **mem_data_out**, **mem_tran**, **reti_tran** and **t1** signals are all inactive (Low) during this time.

t1 (output, active-High) The T1 signal is active during the first clock cycle of a bus transaction. This signal is inactive during the Halt and Sleep states.

tout₀, **tout₁** (outputs, active-High) The Terminal Count Output signals are the terminal-count outputs from the two PRT channels.

tx_{a0}, **tx_{a1}** (outputs, active-High) The Asynchronous Transmit Data signals are the serial data outputs for the two ASCII channels.

txs (output, active-High) The Synchronous Transmit Data signal is the serial data output for the CSIO channel.

wait_req (input, active-High) The Wait Request signal temporarily halts the CPU, usually to wait for memory access time to be met. The wait request is not honored during the bus idle state, or while the **halt_tran** or **sleep_tran** signals are active.

External Timing

The Y90-180 uses a uniform two-clock-cycle machine cycle. This consistent timing simplifies the design of logic external to the CPU makes it easier to track the state of the CPU.

The memory interface timing and signals are designed to make it easy to interface to standard ASIC and FPGA memories. It uses separate read and write strobes.

The I/O interface is very close to the AMBA Peripheral Bus (APB) to allow connection to APB peripherals with a minimum of logic. It uses a single strobe with a separate direction control. The only difference relative to the APB is the setup time for the write data. In the APB the write data is setup one clock before the strobe; in this interface the write data changes coincident with the leading edge of the strobe. In most cases this will not be a problem.

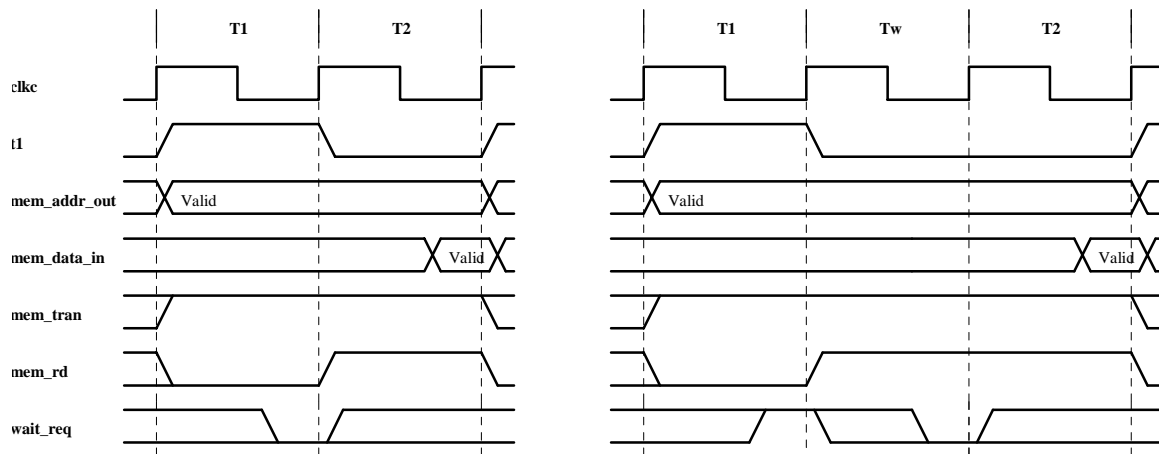
The separate interrupt vector bus provides an easy way to connect to the optional interrupt controller. The interrupt vector bus is used for Mode 0 and Mode 2 maskable interrupts, so if these modes are not used the vector input bus can be tied to ground and the vector strobe output ignored.

In the diagrams below only the relevant signals are shown for each transaction. All other signals are either inactive or hold the previous value. Note that only one of the transaction identifiers (**mem_tran**, **io_tran**, **iack_tran**, **reti_tran**, **halt_tran** and **sleep_tran**) can be active at a time. If all are inactive, an idle bus transaction (usually for address calculation) is in progress. If prefetch is enabled most address calculation idle transactions are replaced by memory transactions. The **dma_ack** signal also indicates that the bus is idle, in response to the **dma_req** signal. The **dma_ack** signal can be active while either **halt_tran** or **sleep_tran** is active.

The **wait_req** input is only sampled for memory, I/O and interrupt acknowledge transactions and is ignored in all other cases. Wait states will disrupt the two-clock-cycle machine cycle rule. If this feature is important but wait states must be used, two wait states per transaction is recommended. If memory access time is an issue it might be better to stretch the first clock cycle of a transaction rather than add Wait states. The uniform two-clock machine cycle makes it relatively straightforward to do this.

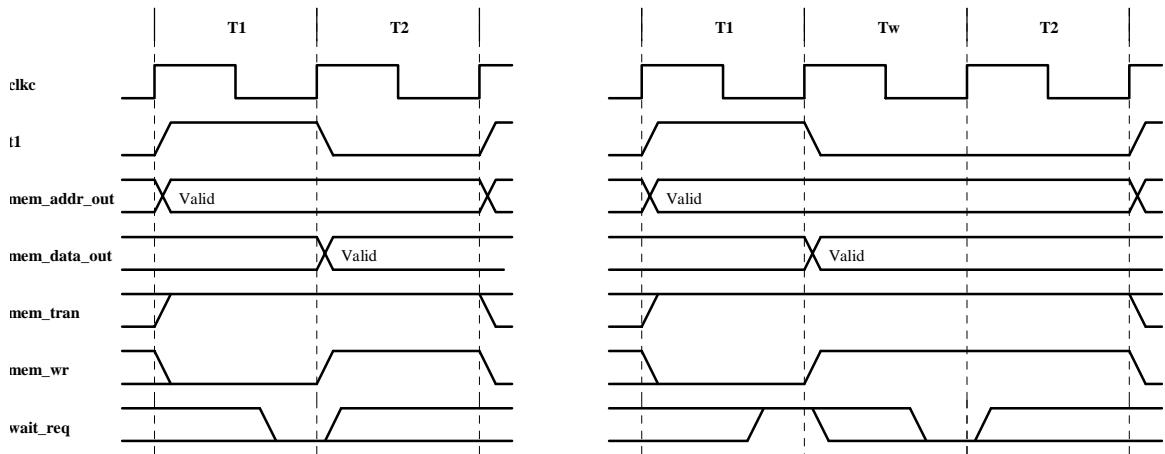
Memory Read

The figure below shows the memory read transaction, without Wait states and with one Wait state. Memory read transactions are used for both instruction and data fetch. There is no separate instruction/data status indicator, although this status exists internally if it is needed.



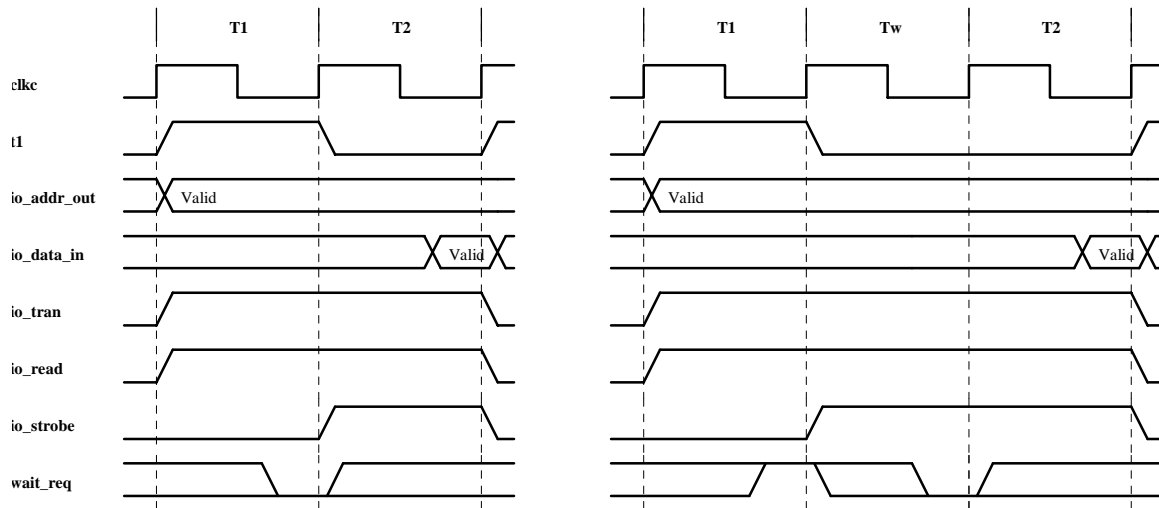
Memory Write

The figure below shows the memory write transaction, without Wait states and with one Wait state.



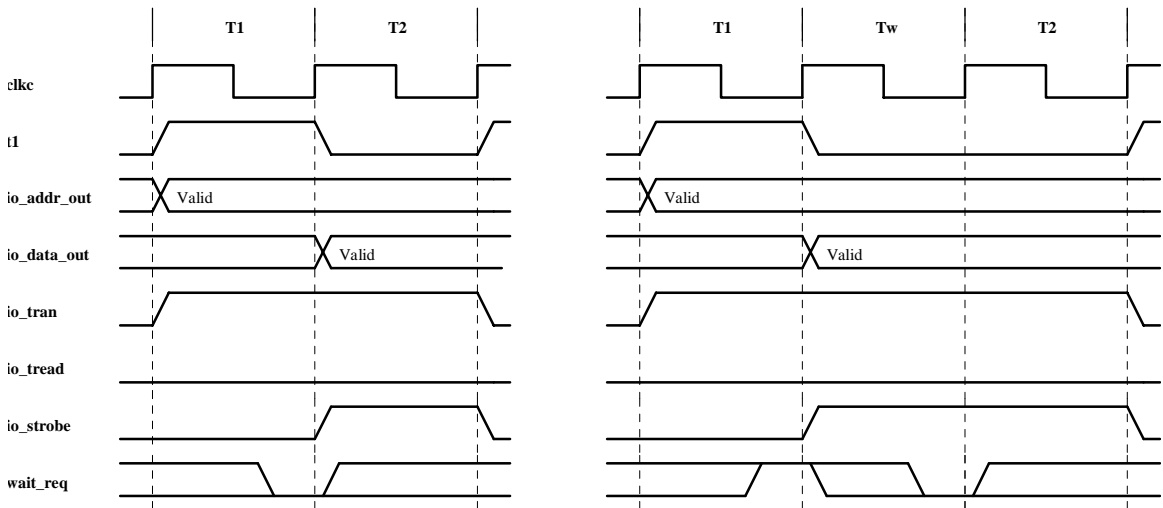
I/O Read

The figure below shows an I/O read transaction, without Wait states and with one Wait state.



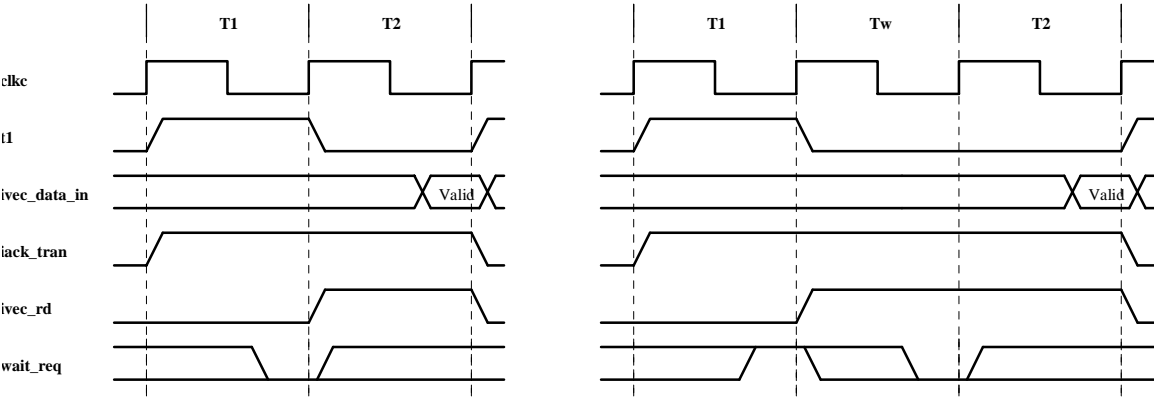
I/O Write

The figure below shows an I/O write transaction, without Wait states and with one Wait state.



Interrupt Acknowledge

The figure below shows the interrupt acknowledge transaction, without Wait states and with one Wait state.



Prefetch

The figure below shows a typical instruction (a memory write) without the prefetch enabled and with the prefetch enabled. The prefetch logic uses address calculation machine cycles to look at the next opcode byte. If this opcode byte is one of the "prefix" bytes (0xCB, 0xDD, 0xED or 0xFD) the logic buffers this byte and will not re-fetch it when the current instruction completes. Only these four prefix bytes will be buffered, even though there are other multi-byte opcodes. Attempting to prefetch for every multi-byte opcode would be significantly more complicated, with marginal performance improvement.

In practice, enabling the prefetch can improve execution time by about 5%, although this obviously depends on the exact code being executed. If Wait states are being used prefetch may not provide any performance gain, because of the Wait states added when prefetching bytes that may later be discarded.

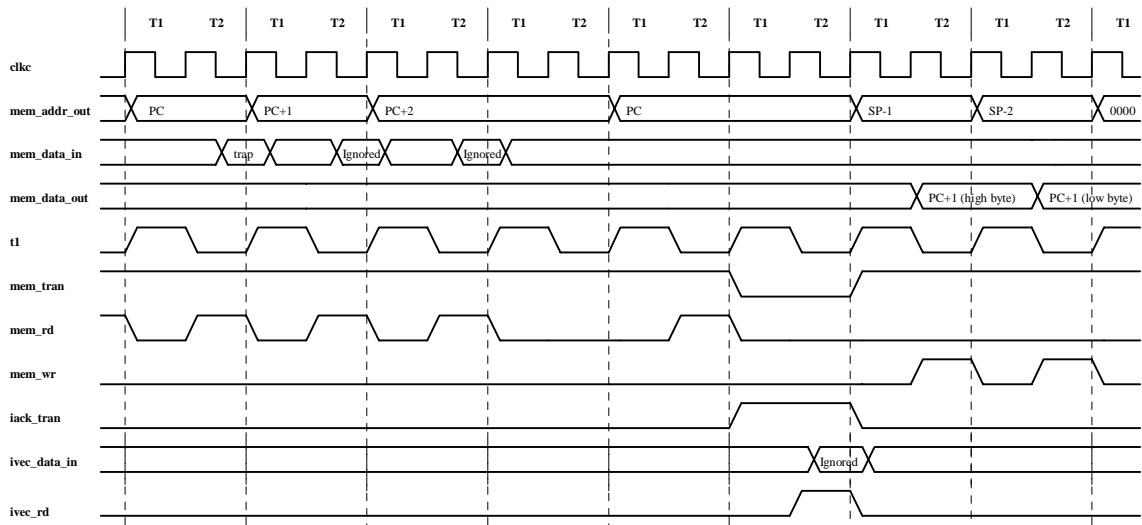
Note that even though an instruction may execute faster when the prefetch is enabled, this instruction will still complete at the same time. However, the next instruction (the one with the first byte prefetched) will complete earlier. This is shown in the diagrams below.

The prefetch can be enabled and disabled on the fly, because the **en_prftch** signal is sampled during the **t1** time of the fetch of the first byte of an instruction.



Illegal Instruction (2nd byte) Trap

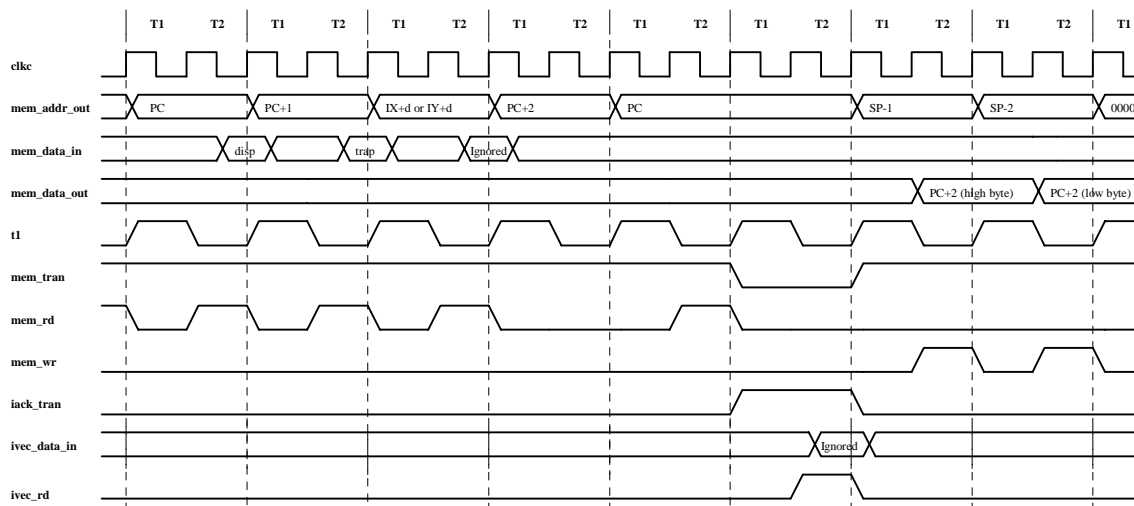
The timing of an undefined second byte opcode trap is shown below. The fetch of the undefined opcode is followed by three machine cycles that flush the pipeline and rewind the Program Counter, an interrupt acknowledge cycle, and two writes to push the PC of the undefined opcode to the stack. The processor then jumps to location 0x0000 and starts fetching instructions.



Information about the trap is latched in the INT/TRAP Control Register. The start of the illegal instruction in this case is the stacked PC value minus one.

Illegal Instruction (3rd byte) Trap

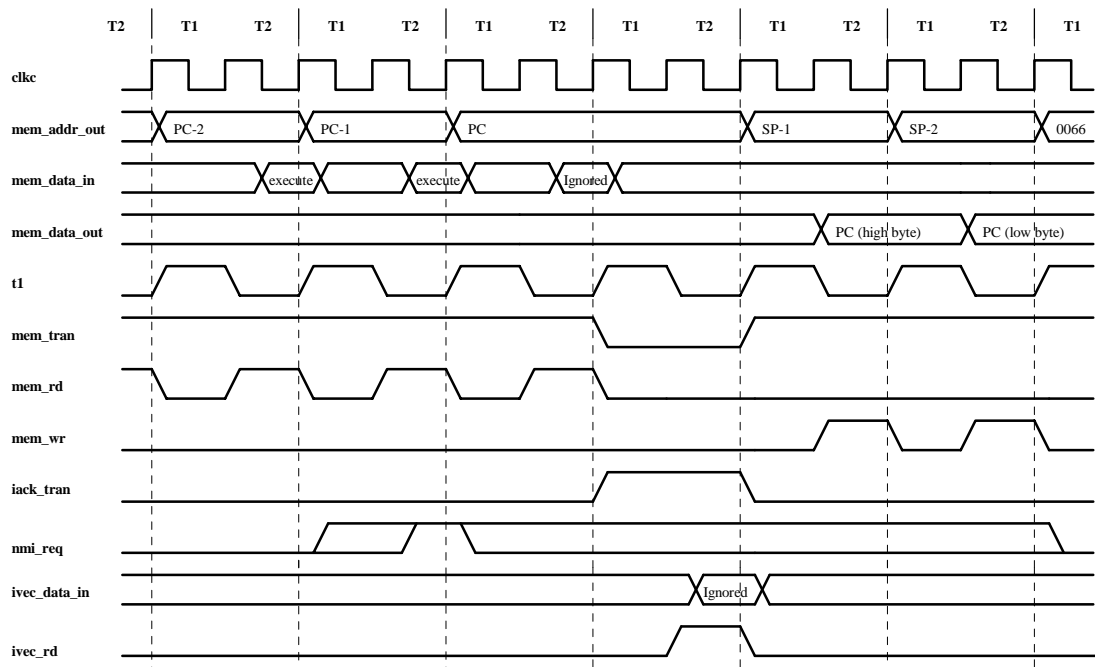
The timing of an undefined third byte opcode trap is shown below. The fetch of the undefined opcode is followed by the normal Read cycle (all three-byte instructions use indexed addressing), two machine cycles that flush the pipeline and rewind the Program Counter, an interrupt acknowledge cycle, and two writes to push the PC of the undefined opcode to the stack. The processor then jumps to location 0x0000 and starts fetching instructions.



Information about the trap is latched in the INT/TRAP Control Register. The start of the illegal instruction in this case is the stacked PC value minus two.

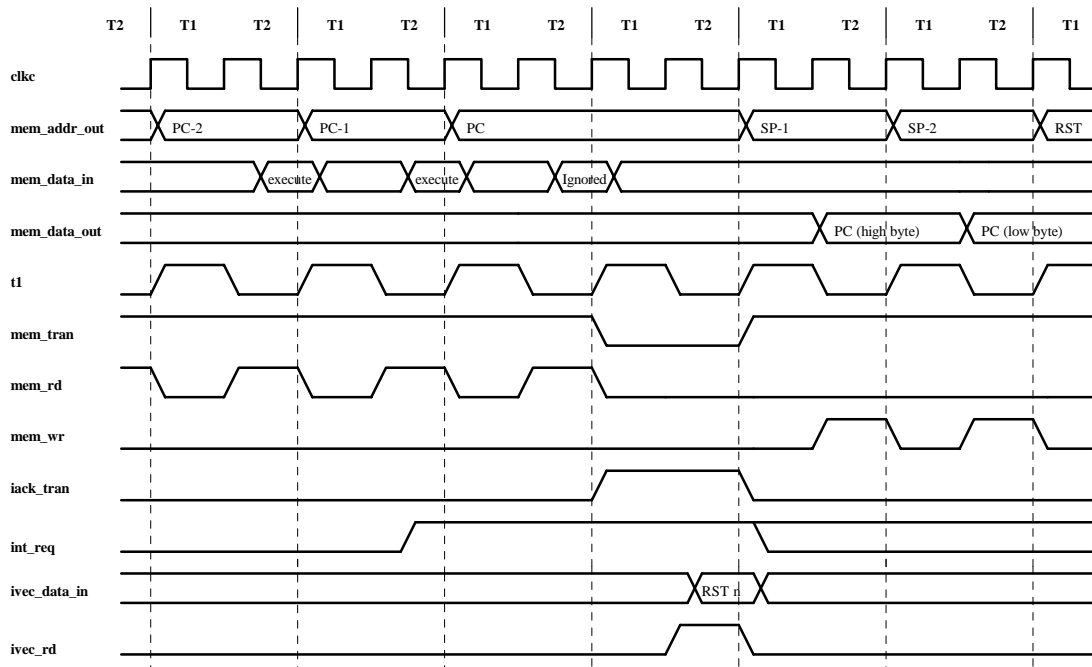
Non-maskable Interrupt

The timing of a non-maskable interrupt acknowledge transaction is shown below. The **nmi_req** input cannot be masked by software. This input must be sampled active by a rising edge of **clk** to be recognized by the processor, but does not need to remain active until the interrupt acknowledge transaction. In fact, to prevent an endless loop of acknowledges, the **nmi_req** input must be de-asserted before the start of the fetch of the first instruction of the service routine. The acknowledge sequence consists of an aborted instruction fetch, the interrupt acknowledge, and two writes to push the contents of the program counter onto the stack. Execution then begins at 0x0066 with an instruction fetch. The non-maskable interrupt service routine must end with the RETN instruction to properly restore the state of the interrupt enable flag prior to the non-maskable interrupt.



Interrupt Mode 0

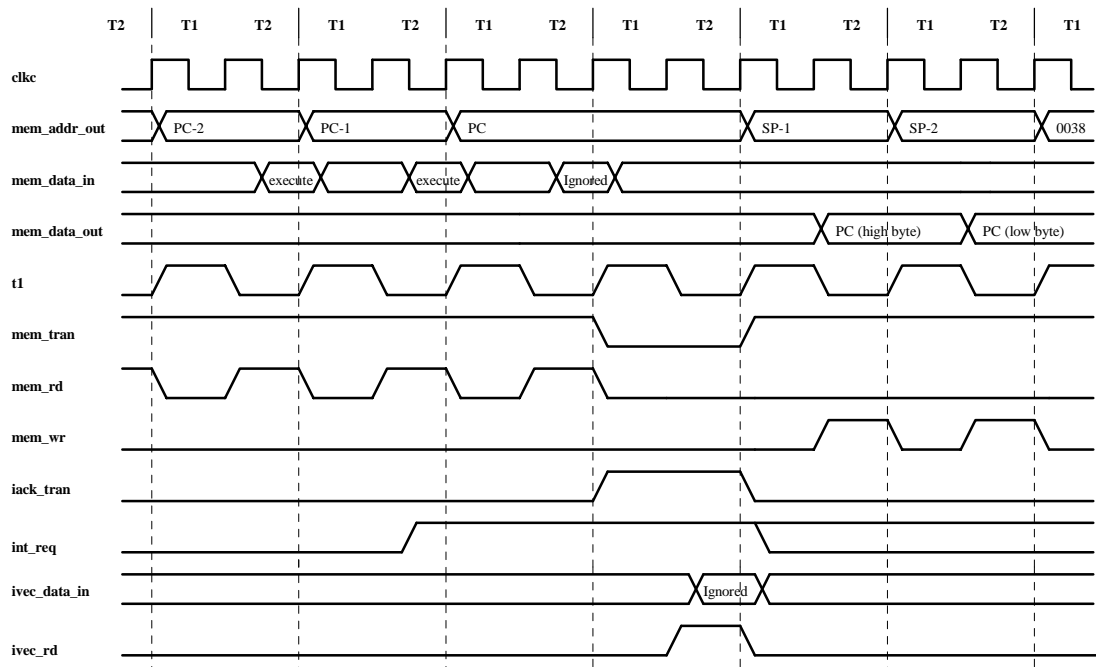
The timing of a Mode 0 maskable interrupt acknowledge is shown below. The **int0_req** signal needs to remain active until the interrupt acknowledge transaction. The acknowledge sequence consists of an aborted instruction fetch, the interrupt acknowledge, and two writes to push the contents of the program counter onto the stack. Execution then begins at the restart address specified by the RST instruction fetched during the interrupt acknowledge. Execution then begins at the restart address specified by the RST instruction fetched during the interrupt acknowledge with an instruction fetch.



The use of an RST instruction is enforced by the hardware, which only uses bits 5-3 of the **ivec_data_in** bus to decode one of the eight possible RST instructions.

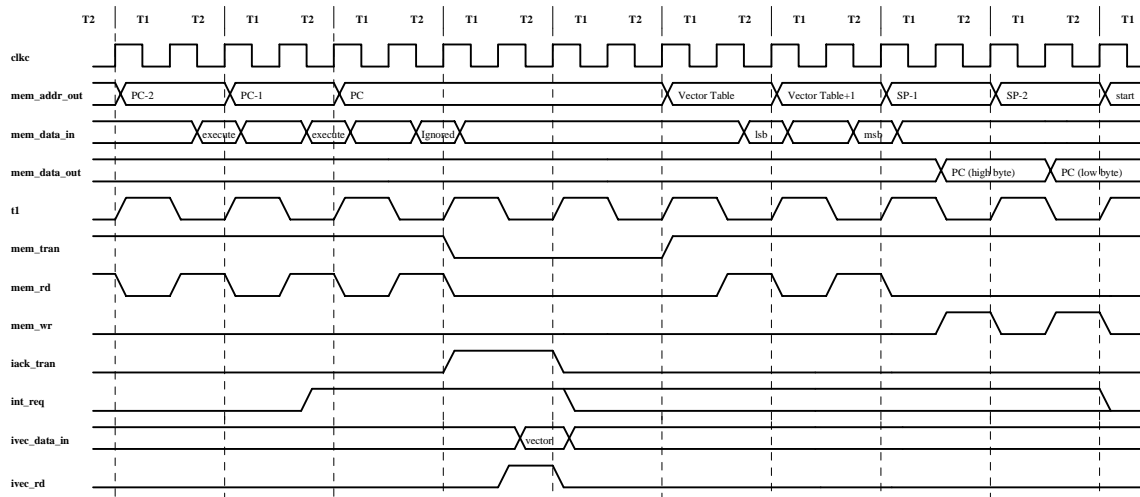
Interrupt Mode 1

The timing of a Mode 1 interrupt acknowledge cycle is shown below. The **int0_req** input needs to remain active until the interrupt acknowledge transaction. The acknowledge sequence consists of an aborted instruction fetch, the interrupt acknowledge, and two writes to push the contents of the program counter onto the stack. Execution then begins at address 0x0038 with an instruction fetch.



Interrupt Mode 2

The timing of a Mode 2 maskable interrupt acknowledge is shown below. The **int0_req**, **int1_req** or **int2_req** input needs to remain active until the interrupt acknowledge transaction. The acknowledge sequence consists of an aborted instruction fetch, the interrupt acknowledge, an address calculation cycle, two reads of the interrupt vector table and two writes to push the contents of the program counter onto the stack. The processor automatically jumps to the address fetched from the interrupt vector table for the service routine. The upper eight bits of the interrupt vector 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.

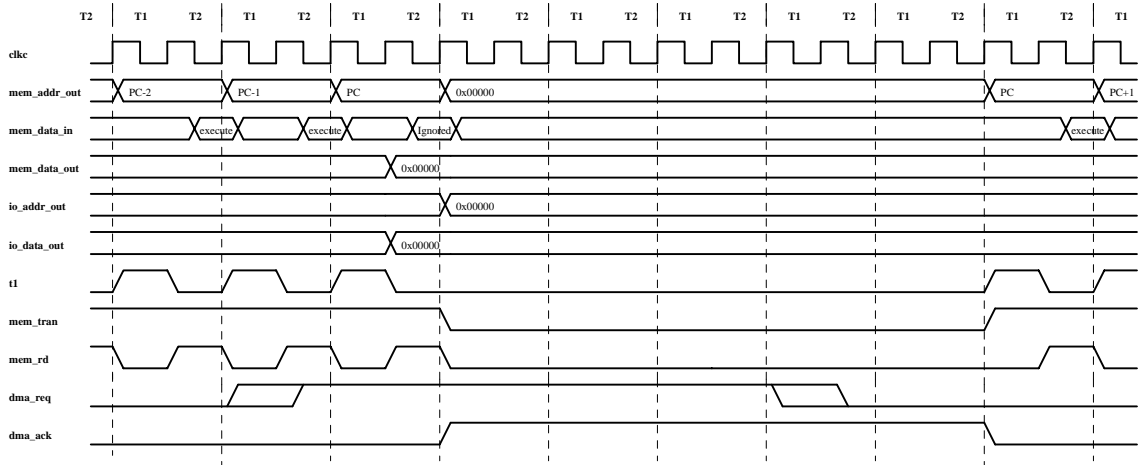


DMA Request/Acknowledge

The timing of a DMA request and acknowledge is shown below. Note that like an interrupt, the **dma_req** signal is only sampled at the end of instructions. This guarantees that all instructions are atomic.

The delay from the **dma_req** signal to the **dma_ack** signal is always at least one bus cycle, irrespective of whether the processor is running, in the Halt state or in the Sleep state. This implies that it is more efficient to transfer multiple bytes each time that the **dma_req** signal is activated.

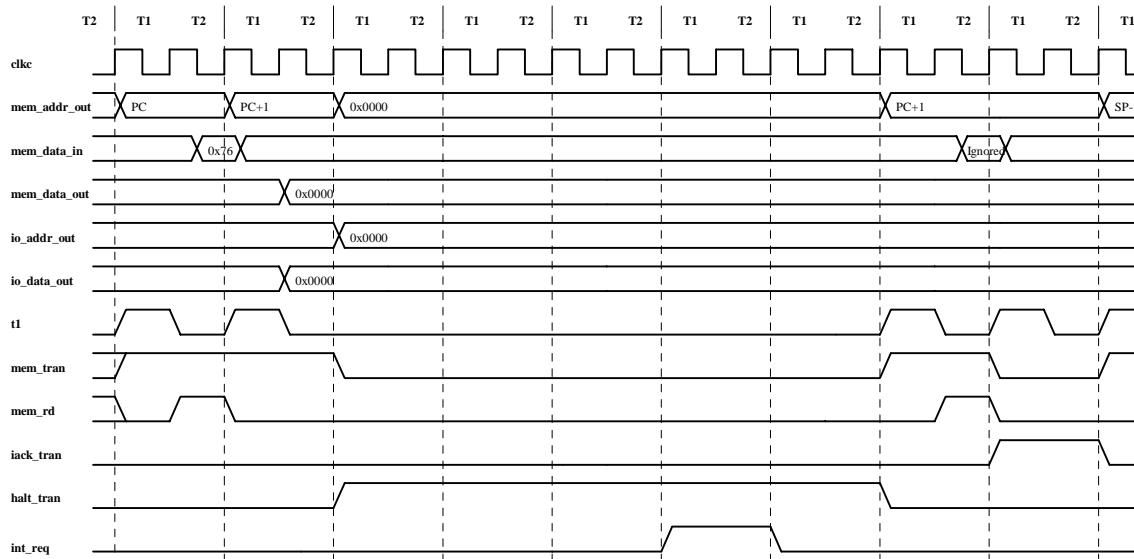
The **dma_req** signal can be asserted during the Halt or Sleep states. In this case the active **dma_req** signal will take precedence over **int_req** or **nmi_req** and inhibit either of these signals from causing an exit from the Halt or Sleep state. Once the **dma_req** signal is deasserted any pending or future interrupt request will cause the exit from the Halt or Sleep state.



Halt state

The Halt state is entered when the HALT instruction is executed, as shown below. In the Halt state the processor freezes, for an unlimited number of two clock cycle machine cycles, with the **halt_tran** output active. The only way to exit the Halt state is with either an interrupt (either **nmi_req**, **int_req** or an internal interrupt) or via reset. Note that **int_req** or an internal interrupt can only be used to exit the Halt mode if interrupts are enabled when the HALT instruction is executed. The timing for exiting the Halt state with an interrupt is also shown below.

If the Halt state is exited by an interrupt, the processor will resume instruction execution (after the interrupt service routine) at the address of the instruction following the HALT instruction. The minimum width of the **halt_tran** signal is two clock cycles.



The Halt state in this design is slightly different from that in the Z80 or Z180 microprocessors. In those designs the processor continues to fetch the Halt instruction during the Halt state, leading to continued power dissipation. Since this operation requires the special step of “rewinding” the PC, no attempt was made to match this operation. Rather, the Halt state and the Sleep state are essentially identical, reducing the power consumption to a minimum by minimizing the number of signals that are transitioning during these states.

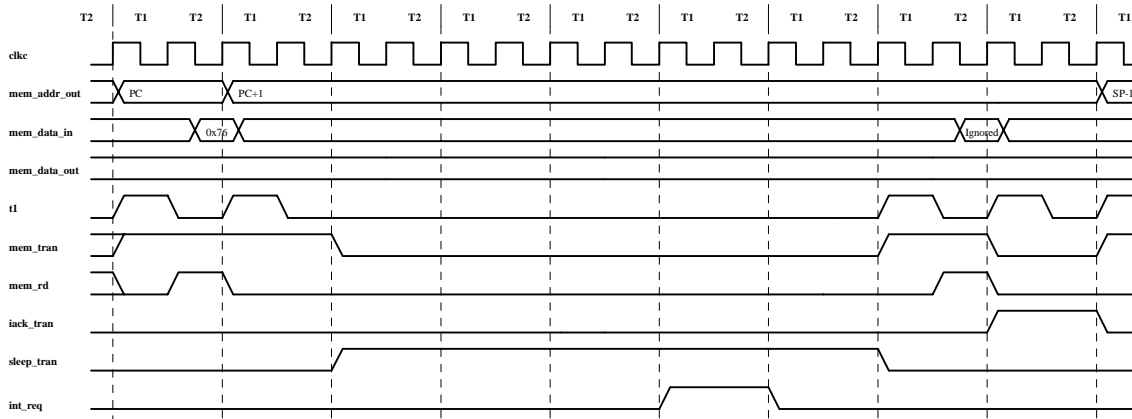
The Halt state differs from the Sleep state only for the case where interrupts are disabled. In the Halt state, if interrupts are disabled only the **nmi_req** or a reset (from any of the

various sources) will cause an exit from this state. In the Sleep state if interrupts are disabled a rising edge on the **int_req** will force an exit from the Sleep state, with execution continuing with the instruction following the SLP instruction.

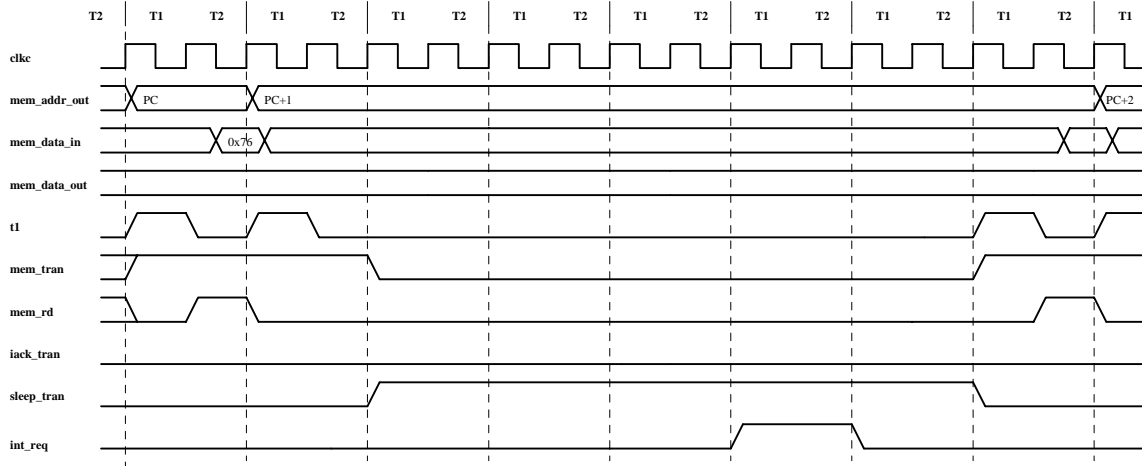
Sleep state

The Sleep state is entered when the SLP instruction is executed, as shown below. In the Sleep state the processor freezes, for an unlimited number of two clock cycle machine cycles, with the **sleep_tran** output active. The only way to exit the Sleep state is with either an interrupt (either **nmi_req**, **int_req** or an internal interrupt) or via reset. The **int_req** signal or an internal interrupt can be used to exit the Sleep mode irrespective of whether or not interrupts are enabled when the SLP instruction is executed.

The timing for exiting the Sleep state with an enabled interrupt or non-maskable interrupt is shown below. In this case the processor will resume instruction execution (after the interrupt service routine) at the address of the instruction following the SLP instruction.



In the case where the Sleep state exit is caused by a maskable interrupt while interrupts are disabled the processor merely resumes execution at the address of the instruction following the SLP instruction, without going through an interrupt service routine. Note that the minimum width of the **sleep_tran** signal is two clock cycles.



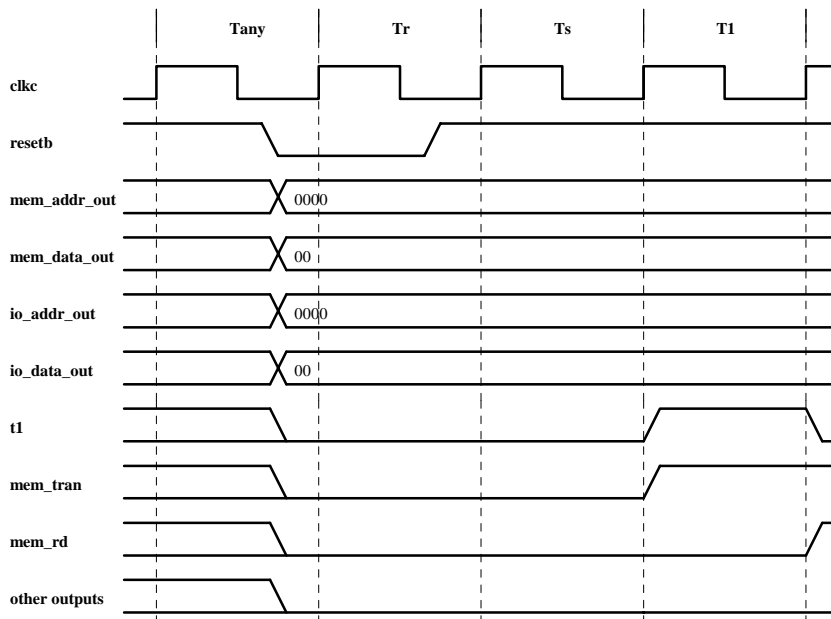
Reset

The Reset state is entered immediately when the **resetb** signal goes Low, independent of the current state, and this state continues until the first rising edge of **clk** after the **resetb** signal is de-asserted. At this rising edge there is a one clock cycle transient state to set up the internal pipeline controls, and on the next clock the processor begins fetching the first instruction from address 0x0000.

Software starting at location 0x0000 must be able to distinguish between reset, execution of an RST 0 instruction or a trap. All of these cases cause the Program Counter to be reset to 0x0000.

The minimum width of the **resetb** signal is set by the flip-flops used in the design. The setup time for the **resetb** signal to the rising edge of the **clk** signal is likewise determined by the flip-flops used in the design.

The **clearb** signal has the same timing requirements as the **resetb** signal. The **clearb** signal should only be used in the power-on case, and only affects those flip-flops not affected by the **resetb** signal.



Instruction Set

This chapter presents the assembly language syntax, addressing modes, flag settings, binary encoding, and execution time for the Y90-180 instruction set. The entire instruction set is presented in alphabetical order.

The assembly language syntax is identical to that used by the original Zilog assembler. Different assembler programs may or may not use identical syntax. The syntax is presented generically at the beginning of each instruction, with the details presented for each addressing mode later in each entry.

The operation of each instruction is specified in a format similar to Verilog HDL for minimum ambiguity, but no descriptive text or examples are included.

The effect of the instruction on each flag is listed, with a brief description. Normally the flags are updated by the main operation of the instruction, but for some complex instructions different flags may be affected by different parts of the instruction. This is specified in the description. The flags are organized as below in the F (Flag) register:

S	Z	U5	H	U3	P/V	N	C
---	---	----	---	----	-----	---	---

These flags have the following meanings:

Flag	Meaning
S	Sign (a copy of the MSB of the result).
Z	Zero (indicating that the result was zero).
U5	Unused Bit 5 (an unused Flag register bit).
H	Half-Carry (carry out of the lower nibble, used for BCD math).
U3	Unused Bit 3 (an unused Flag register bit).
P/V	Parity/Overflow (parity of the result, or arithmetic overflow; depends on the instruction)
N	Negative (add/subtract flag, necessary for BCD math)
C	Carry (arithmetic carry, or shift linkage bit)

Fields in the instruction are listed using shortcuts for common fields. These shortcuts should be self-explanatory in most cases, but will be detailed here for completeness.

The most common field in the instruction specifies a CPU register, employing the following encoding:

rrr	Register Selected
000	B
001	C
010	D
011	E
100	H
101	L
111	A (Accumulator)

Word registers are similarly encoded, although the exact encoding depends on the instruction:

dd, ss, tt, xx or yy	dd, ss Register	tt Register	xx Register	yy Register
00	BC	BC	BC	BC
01	DE	DE	DE	DE
10	HL	HL	IX	IY
11	SP	AF	SP	SP

The execution time for instructions is always a multiple of two clocks. Any number in parentheses is the execution time when the prefetch is enabled, via the **en_prftch** signal into the core. When enabled, the prefetch operation uses any address calculation time to look at the first byte of the next instruction. If this instruction byte can be pre-decoded the byte will be buffered for use when the current instruction finishes. This results in the execution time in parentheses. Only instructions that require more than one machine cycle to execute can be pre-decoded.

ADC

Add With Carry

ADC A, src

src: R, IM, IR, X

Operation: $A \leftarrow A + \text{src} + \text{CF}$

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic carry out of bit 3; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Cleared.
- C:** Set if arithmetic carry out of bit 7; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	ADC A, r	10001rrr	2
IM:	ADC A, n	11001110 ----n---	4
IR:	ADC A, (HL)	10001110	6 (4)
X:	ADC A, (IX+d) or ADC A, (IY+d)	11y11101 10001110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

ADC

Add With Carry (Word)

ADC HL, src

src: RR

Operation: HL <= HL + src + CF

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic carry out of bit 11; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Cleared.
- C:** Set if arithmetic carry out of bit 15; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
RR:	ADC HL, ss	<table border="1"><tr><td>11101101</td></tr><tr><td>01ss1010</td></tr></table>	11101101	01ss1010	4
11101101					
01ss1010					

Notes:

1. The **ss** field uses the standard word register encoding.

ADD

Add

ADD A, src src: R, IM, IR, X

Operation: $A \leq A + \text{src}$

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic carry out of bit 3; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Cleared.
- C:** Set if arithmetic carry out of bit 7; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	ADD A, r	1000rrr	2
IM:	ADD A, n	11000110 ----n---	4
IR:	ADD A, (HL)	10000110	6 (4)
X:	ADD A, (IX+d) or ADD A, (IY+d)	11y11101 10000110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY

ADD

Add (Word)

ADC dst, src

dst: HL, IX, IY

src: RR

Operation: dst <= dst + src

Flags: **S:** Unaffected.
Z: Unaffected.
H: Set if arithmetic carry out of bit 11; cleared otherwise.
P/V: Unaffected.
N: Cleared.
C: Set if arithmetic carry out of bit 15; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks
RR:	ADD HL, ss	00ss1001	2
	ADC IX, xx	11011101	4
		01xx1001	
	ADC IY, yy	11111101	4
		01yy1001	

Notes:

1. The **ss**, **xx** and **yy** fields use the standard word register select encodings.

AND

Logical AND

AND A, src

src: R, IM, IR, X

Operation: A <= A & src

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set.
- P/V:** Set if parity of result even; cleared otherwise.
- N:** Cleared.
- C:** Cleared.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	AND A, r	10100rrr	2
IM:	AND A, n	11100110 ----n---	4
IR:	AND A, (HL)	10100110	6 (4)
X:	AND A, (IX+d) or AND A, (IY+d)	11y11101 10100110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY

BIT

Bit Test

BIT b, src src: R, IR, X

Operation: $Z \leftarrow \sim \text{src}[b]$

Flags: **S:** Unaffected.
Z: Set if tested bit is zero; cleared otherwise.
H: Set.
P/V: Unaffected.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	BIT b, r	<table border="1"><tr><td>11001011</td></tr><tr><td>01bbrrr</td></tr></table>	11001011	01bbrrr	4		
11001011							
01bbrrr							
IR:	BIT b, (HL)	<table border="1"><tr><td>10100110</td></tr><tr><td>01bbb1110</td></tr></table>	10100110	01bbb1110	8 (6)		
10100110							
01bbb1110							
X:	BIT b, (IX+d) or BIT b, (IY+d)	<table border="1"><tr><td>11y11101</td></tr><tr><td>11001011</td></tr><tr><td>----d---</td></tr><tr><td>01bbb110</td></tr></table>	11y11101	11001011	----d---	01bbb110	10
11y11101							
11001011							
----d---							
01bbb110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.
3. The **bbb** field uses normal binary encoding.
4. For the original Z80, the **S** and **C** flags are undefined.

CALL

Call Subroutine

CALL dst

dst: DA

Operation: SP \leq SP - 2
 (SP) \leq PC
 PC \leq dst

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks			
DA:	CALL mn	<table border="1"> <tr> <td>11001101</td> </tr> <tr> <td>----n---</td> </tr> <tr> <td>----m---</td> </tr> </table>	11001101	----n---	----m---	10
11001101						
----n---						
----m---						

CALL

Conditional Call Subroutine

CALL cc, dst

dst: DA

Operation: if (cc = true) begin
 SP <= SP - 2
 (SP) <= PC
 PC <= dst
end

Flags: S: Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks			
DA:	CALL cc, mn	<table border="1"><tr><td>11fff100</td></tr><tr><td>----n---</td></tr><tr><td>----m---</td></tr></table>	11fff100	----n---	----m---	10/6 (taken/not taken)
11fff100						
----n---						
----m---						

Notes:

1.	Mnemonic	Encoding (fff)	Meaning	Flag case
	NZ	000	Non-zero	Z = 0
	Z	001	Zero	Z = 1
	NC	010	Non-carry	C = 0
	C	011	Carry	C = 1
	PO	100	Parity Odd	P/V = 0
	PE	101	Parity Even	P/V = 1
	P	110	Plus	S = 0
	M	111	Minus	S = 1

CCF

Complement Carry Flag

CCF

Operation: $CF \leq \sim CF$

Flags:

- S:** Unaffected.
- Z:** Unaffected.
- H:** Copy of previous value of Carry flag.
- P/V:** Unaffected.
- N:** Cleared.
- C:** Set if previous Carry flag was zero; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks
IM:	CCF	00111111	2

Notes:

1. The default operation of the **H** flag for this instruction matches that of the original Z80 CPU. The original Z180 CPU behaves differently, clearing the **H** flag for this instruction. To enable Z180 compatibility, use the 'define Z180_CCF' option in the Verilog source code file version.v.

CP

Compare

CP A, src

src: R, IM, IR, X

Operation: A - src

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic borrow out of bit 3; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Set.
- C:** Set if arithmetic borrow out of bit 7; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	CP A, r	1011rrr	2
IM:	CP A, n	11111110 ----n---	4
IR:	CP A, (HL)	10111110	6 (4)
X:	CP A, (IX+d) or CP (IY+d)	11y11101 10111110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

CPD

Compare and Decrement

CPD

Operation: A - (HL)
HL \leq HL - 1
BC \leq BC - 1

Flags: **S:** Set if result of compare is negative, cleared otherwise.
Z: Set if result of compare is zero; cleared otherwise.
H: Set if arithmetic borrow out of bit 3 during compare; cleared otherwise.
P/V: Set if result of BC decrement is non-zero; cleared otherwise.
N: Set.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	CPD	<table border="1"><tr><td>11101101</td></tr><tr><td>10101001</td></tr></table>	11101101	10101001	10 (8)
11101101					
10101001					

CPDR

Compare, Decrement and Repeat

CPDR

Operation: A - (HL)
 HL <= HL - 1
 BC <= BC - 1
 repeat if BC != 0 and A - (HL) != 0

Flags: **S:** Set if result of compare is negative, cleared otherwise.
 Z: Set if result of compare is zero; cleared otherwise.
 H: Set if arithmetic borrow out of bit 3 during compare; cleared otherwise.
 P/V: Set if result of BC decrement is non-zero; cleared otherwise.
 N: Set.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	CPDR	<table border="1"><tr><td>11101101</td></tr><tr><td>10111001</td></tr></table>	11101101	10111001	8 + 4i
11101101					
10111001					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine.
2. Interrupts are sampled during each memory read operation.

CPI

Compare and Increment

CPI

Operation: A - (HL)
 HL <= HL + 1
 BC <= BC - 1

Flags: **S:** Set if result of compare is negative, cleared otherwise.
 Z: Set if result of compare is zero; cleared otherwise.
 H: Set if arithmetic borrow out of bit 3 during compare; cleared otherwise.
 P/V: Set if result of decrementing BC is non-zero; cleared otherwise.
 N: Set.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	CPI	<table border="1"><tr><td>11101101</td></tr><tr><td>10100001</td></tr></table>	11101101	10100001	10 (8)
11101101					
10100001					

CPIR

Compare, Increment and Repeat

CPIR

Operation: A - (HL)
 HL <= HL + 1
 BC <= BC - 1
 repeat if BC != 0 and A - (HL) != 0

Flags: **S:** Set if result of compare is negative, cleared otherwise.
 Z: Set if result of compare is zero; cleared otherwise.
 H: Set if arithmetic borrow out of bit 3 during compare; cleared otherwise.
 P/V: Set if result of decrementing BC is non-zero; cleared otherwise.
 N: Set.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	CPIR	<table border="1"><tr><td>11101101</td></tr><tr><td>10110001</td></tr></table>	11101101	10110001	8 + 4i
11101101					
10110001					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each memory read operation.

CPL

Complement

CPL

Operation: $A \leftarrow \sim A$

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Set.
 P/V: Unaffected.
 N: Set.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	CPL	00101111	2

DAA

Decimal Adjust Accumulator

DAA

Operation: A <= Decimal Adjust A

Flags: **S:** Set if result is negative; cleared otherwise.
Z: Set if result is zero; cleared otherwise.
H: See table below.
P/V: Set if result has even parity; cleared otherwise.
N: Unaffected.
C: See table below.

Addressing Modes	Assembly Syntax	Encoding	Clocks	
	DAA	<table border="1"><tr><td>00100111</td></tr></table>	00100111	2
00100111				

Notes:

Instruction	C before DAA	A[7:4] before DAA	H before DAA	A[3:0] before DAA	Number added to A	C after DAA
ADC, ADD or INC	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
DEC, NEG, SUB or SBC	1	0-3	1	0-3	66	1
	0	0-9	0	0-9	00	0
	0	0-8	1	6-F	FA	0
	1	7-F	0	0-9	A0	1
	1	6-F	1	6-F	9A	1

DI

Disable Interrupt

DI

Operation: IFF1 <= 0
 IFF2 <= 0

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
DI		11110011	2

Notes:

1. Interrupts are last sampled during the machine cycle that fetches this instruction.

DJNZ

Decrement, Jump if Non-zero

DJNZ e

Operation: B \leq B - 1
if (B \neq 0) PC \leq PC + e (where PC is the PC of this instruction)

Flags: S: Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	DJNZ e	<table border="1"><tr><td>00010000</td></tr><tr><td>--(e-2)-</td></tr></table>	00010000	--(e-2)-	6
00010000					
--(e-2)-					

Notes:

1. Relative to the address of this instruction, the jump range is -126 to +129. Relative to the address of the next instruction, the jump range is -128 to +127.

EI

Enable Interrupt

EI

Operation: IFF1 <= 1
 IFF2 <= 1

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
EI		11111011	2

Notes:

1. Interrupts are first sampled during the fetch of the next instruction. If an interrupt is pending this instruction fetch will be ignored and an interrupt acknowledge cycle started.

EX

Exchange with Top-of-Stack

EX (SP), src

src: HL, IX, IY

Operation: (SP) <=> L or IXL or IYL
(SP+1) <=> H or IXH or IYH

Flags: S: Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	EX (SP), HL	11100011	12 (10)
	EX (SP), IX or EX (SP), IY	11y11101 11100011	14 (12)

Notes:

1. y = 0 selects IX and y = 1 selects IY

EX AF, AF'

Exchange Accumulator

EX AF, AF'

Operation: AF <=> AF'

Flags: **S:** Replaced by alternate flag.
Z: Replaced by alternate flag.
H: Replaced by alternate flag.
P/V: Replaced by alternate flag.
N: Replaced by alternate flag.
C: Replaced by alternate flag.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	EX AF, AF'	00001000	2

Notes:

1. No data is actually moved. Instead the registers are renamed.

EX

Exchange (Word)

EX DE, HL

Operation: DE <=> HL

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	EX DE, HL	11101011	2

EXX

Exchange Register Bank

EXX

Operation: BC <=> BC'
DE <=> DE'
HL <=> HL'

Flags: S: Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	EXX	11011001	2

Notes:

1. No data is actually moved. Instead the registers are renamed.

HALT

Halt

HALT

Operation: activate Halt signal and wait for interrupt

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	HALT	01110110	4 + 2n

Notes:

1. The CPU halts with an idle bus until an interrupt is requested. The address pushed to the stack during the interrupt acknowledge is the address of the next instruction. During Halt the **mem_addr_bus** and **io_addr_bus** are driven with 0x0000, and the **mem_data_bus** and **io_data_bus** are driven with 0x00.

IM i

Operation: Set Interrupt Mode i

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
IM 0		<table border="1"><tr><td>11101101</td></tr><tr><td>01000110</td></tr></table>	11101101	01000110	4
11101101					
01000110					
IM 1		<table border="1"><tr><td>11101101</td></tr><tr><td>01010110</td></tr></table>	11101101	01010110	4
11101101					
01010110					
IM 2		<table border="1"><tr><td>11101101</td></tr><tr><td>01011110</td></tr></table>	11101101	01011110	4
11101101					
01011110					

Notes:

1. Interrupt Mode 0 expects an RST instruction on the **ivec_bus** during the interrupt acknowledge cycle. Only an RST instruction is allowed.
2. Interrupt Mode 1 always jumps to location 0x0038 in response to a maskable interrupt request.
3. Interrupt Mode 2 uses the interrupt vector returned on the **ivec_bus** during an interrupt acknowledge cycle, along with the contents of the I register, to access an interrupt vector table in memory. The address stored at the selected location in the interrupt vector table is the starting address of the interrupt service routine. Note that the least-significant bit of the interrupt vector must be zero to account for the two-byte entries in the interrupt vector table.

IN

Input

IN A, src

src: DA

Operation: $A \leftarrow I/O(A:n)$

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	IN A, (n)	<table border="1"><tr><td>11011011</td></tr><tr><td>----n---</td></tr></table>	11011011	----n---	8 (6)
11011011					
----n---					

IN

Input

IN r, (C)

dst: R

Operation: r <= I/O(BC)

Flags:

- S:** Set if the input data is negative; cleared otherwise.
- Z:** Set if the input data is zero; cleared otherwise.
- H:** Cleared.
- P/V:** Set if the parity of the input data is even; cleared otherwise.
- N:** Cleared.
- C:** Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	IN r, (C)	<table border="1"><tr><td>11101101</td></tr><tr><td>01rrr000</td></tr></table>	11101101	01rrr000	8 (6)
11101101					
01rrr000					

Notes:

1. The **rrr** field uses the standard register select encoding

IN0

Input (page 0)

IN0 r, (n)

dst: R

Operation: r <= I/O(0:n)

Flags:

- S:** Set if input byte is negative; cleared otherwise.
- Z:** Set if input byte is zero; cleared otherwise.
- H:** Cleared.
- P/V:** Set if parity of input byte is even; cleared otherwise.
- N:** Cleared.
- C:** Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks			
	IN0 r, (n)	<table border="1"><tr><td>11101101</td></tr><tr><td>00rrr000</td></tr><tr><td>----n---</td></tr></table>	11101101	00rrr000	----n---	10 (8)
11101101						
00rrr000						
----n---						

Notes:

1. The **rrr** field uses the standard register select encoding
2. This instruction is not present in the original Z80, but is a feature of the Z180.

INC

Increment

INC dst dst: R, IR, X

Operation: dst <= dst + 1

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic carry out of bit 3; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Cleared.
- C:** Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	INC r	00rrr100	2
IR:	INC (HL)	00110100	8 (6)
X:	INC (IX+d) or INC (IY+d)	11y11101	12 (10)
		00110100	
		----d---	

Notes:

1. The **rrr** field uses the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

INC

Increment (Word)

INC dst

dst: RR, IX, IY

Operation: dst \leq dst + 1

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
RR:	INC ss	00dd0011	2
IX, IY	INC IX or INC IY	11y11101	4
		00100011	

Notes:

1. The **dd** field uses the standard word register encoding.

IND

Input and Decrement

IND

Operation: (HL) <= I/O(BC)
B <= B - 1
HL <= HL -1

Flags: **S:** Unaffected.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Unaffected.
P/V: Unaffected.
N: Set.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	IND	<table border="1"><tr><td>11101101</td></tr><tr><td>10101010</td></tr></table>	11101101	10101010	10 (8)
11101101					
10101010					

Notes:

1. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

INDR

Input, Decrement and Repeat

INDR

Operation: (HL) <= I/O(BC)
B <= B - 1
HL <= HL - 1
repeat if B != 0

Flags: **S:** Unaffected.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Unaffected.
P/V: Unaffected.
N: Set.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	INDR	<table border="1"><tr><td>11101101</td></tr><tr><td>10111010</td></tr></table>	11101101	10111010	8 + 4i
11101101					
10111010					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each I/O read operation.
3. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

INI

Input and Increment

INI

Operation: (HL) <= I/O(BC)
B <= B - 1
HL <= HL + 1

Flags: **S:** Unaffected.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Unaffected.
P/V: Unaffected.
N: Set.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	INI	<table border="1"><tr><td>11101101</td></tr><tr><td>10100010</td></tr></table>	11101101	10100010	10 (8)
11101101					
10100010					

Notes:

1. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

INIR

Input, Increment and Repeat

INIR

Operation: (HL) <= I/O(BC)
B <= B - 1
HL <= HL + 1
repeat if B != 0

Flags: **S:** Unaffected.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Unaffected.
P/V: Unaffected.
N: Set.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
INIR		<table border="1"><tr><td>11101101</td></tr><tr><td>10110010</td></tr></table>	11101101	10110010	8 + 6i
11101101					
10110010					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each I/O read operation.
3. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

JP

Jump

JP dst dst: IM, IR

Operation: PC <= dst

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
IR:	JP (HL)	11101001	4
	JP (IX) or JP (IY)	11y11101 11101001	6
IM:	JP mn	11000011	8
		----n---	
		----m---	

Notes:

1. The indirect jumps use the contents of the register directly for the jump address.

JP

Conditional Jump

JP cc, mn

Operation: if (cc = true) PC <= mn

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
IM:	JP cc, mn	11fff010	8 (taken)
		----n---	6 (not taken)
		----m---	

Notes:

1.	Mnemonic	Encoding (fff)	Meaning	Flag case
	NZ	000	Non-zero	Z = 0
	Z	001	Zero	Z = 1
	NC	010	Non-carry	C = 0
	C	011	Carry	C = 1
	PO	100	Parity Odd	P/V = 0
	PE	101	Parity Even	P/V = 1
	P	110	Plus	S = 0
	M	111	Minus	S = 1

JR

Jump Relative

JR e

Operation: PC \leq PC + e (where PC is the PC of this instruction)

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	JR e	<table border="1"><tr><td>00011000</td></tr><tr><td>--(e-2)-</td></tr></table>	00011000	--(e-2)-	6
00011000					
--(e-2)-					

Notes:

1. Relative to the address of this instruction, the jump range is -126 to +129. Relative to the address of the next instruction, the jump range is -128 to +127.

JR

Conditional Jump Relative

JR cc, e

Operation: if (cc = true) PC <= PC + e (where PC is the PC of this instruction)

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	JR cc, e	<table border="1"><tr><td>001cc000</td></tr><tr><td>--(e-2)-</td></tr></table>	001cc000	--(e-2)-	6 (taken) 4 (not taken)
001cc000					
--(e-2)-					

Notes:

1. Relative to the address of this instruction, the jump range is -126 to +129. Relative to the address of the next instruction, the jump range is -128 to +127.

1.	Mnemonic	Encoding (cc)	Meaning	Flag case
	NZ	00	Non-zero	Z = 0
	Z	01	Zero	Z = 1
	NC	10	Non-carry	C = 0
	C	11	Carry	C = 1

LD

Load Accumulator from Memory

LD A, src

src: DA, IR

Operation: A <= src

Flags: S: Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
DA:	LD A, (mn)	00111010	10 (8)
		----n---	
		----m---	
IR:	LD A, (BC)	00001010	6 (4)
	LD A, (DE)	00011010	6 (4)

LD

Load Accumulator from Special Register

LD A, src src: special register

Operation: A <= src

Flags: **S:** Set if the contents of the Special Register is negative; cleared otherwise.
Z: Set if the contents of the Special Register is zero; cleared otherwise.
H: Cleared.
P/V: Loaded with the contents if the IFF2 interrupt enable flag.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	LD A, I	<table border="1"><tr><td>11101101</td></tr><tr><td>01010111</td></tr></table>	11101101	01010111	4
11101101					
01010111					
	LD A, R	<table border="1"><tr><td>11101101</td></tr><tr><td>01011111</td></tr></table>	11101101	01011111	4
11101101					
01011111					

LD

Load Memory from Accumulator

LD dst, A

dst: DA, IR

Operation: dst <= A

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
DA:	LD (mn), A	00110010	10 (8)
		----n---	
		----m---	
IR:	LD (BC), A	00000010	6 (4)
	LD (DE), A	00010010	6 (4)

LD

Load Memory with Immediate

LD dst, n

dst: IR, X

Operation: dst <= n

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
IR:	LD (HL), n	00110110	6
		----n---	
X:	LD (IX+d), n or LD (IY+d), n	11y11101	10
		00110110	
		----d---	
		----n---	

Notes:

1. **y** = 0 selects IX and **y** = 1 selects IY

LD

Load Memory from Register (Word)

LD (mn), src

src: HL, RR, IX, IY

Operation: (mn) <= src

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
HL:	LD (mn), HL	00100010	12 (10)
		----n---	
		----m---	
RR:	LD (mn), ss	11101101	14 (12)
		01ss0011	
		----n---	
		----m---	
IX, IY:	LD (mn), IX or LD (mn), IY	11y11101	14 (12)
		00100010	
		----n---	
		----m---	

Notes:

1. The **ss** field uses the standard word register encoding.

LD

Load Register

LD r, src

dst: R, IM, IR, X

Operation: r <= src

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	LD rd, rs	01rdrsr	2
IM	LD r, n	00rrr110 ----n---	4
IR:	LD r, (HL)	01rrr110	6 (4)
X:	LD r, (IX+d) or LD r, (IY+d)	11y11101 01rrr110 ----d---	10 (8)

Notes:

1. The **rdr**, **rsr** and **rrr** fields use the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

LD

Load Register Immediate (Word)

LD dst, mn

dst: RR, IX, IY

Operation: dst <= mn

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
IM:	LD dd, mn	00dd0001	6
		----n---	
		----m---	
	LD IX, mn or LD IY, mn	11y11101	8
		00100001	
		----n---	
		----m---	

Notes:

1. The **dd** field uses the standard word register encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY

LD

Load Register (Word)

LD dst, (mn)

dst: RR, IX, IY

Operation: dst <= (mn)

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks			
DA:	LD HL, (mn)	<table border="1"><tr><td>00101010</td></tr><tr><td>----n---</td></tr><tr><td>----m---</td></tr></table>	00101010	----n---	----m---	12 (10)
	00101010					
	----n---					
----m---						
LD dd, (mn)	<table border="1"><tr><td>11101101</td></tr><tr><td>01dd1011</td></tr><tr><td>----n---</td></tr><tr><td>----m---</td></tr></table>	11101101	01dd1011	----n---	----m---	14 (12)
11101101						
01dd1011						
----n---						
----m---						
LD IX, (mn) or LD IY, (mn)	<table border="1"><tr><td>11y11101</td></tr><tr><td>00101010</td></tr><tr><td>----n---</td></tr><tr><td>----m---</td></tr></table>	11y11101	00101010	----n---	----m---	14 (12)
11y11101						
00101010						
----n---						
----m---						

Notes:

1. The **dd** field uses the standard word register encoding.

LD

Load Special Register from Accumulator

LD dst, A

dst: special register

Operation: dst <= A

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	LD I, A	<table border="1"><tr><td>11101101</td></tr><tr><td>01000111</td></tr></table>	11101101	01000111	4
11101101					
01000111					
	LD R, A	<table border="1"><tr><td>11101101</td></tr><tr><td>01001111</td></tr></table>	11101101	01001111	4
11101101					
01001111					

LD

Load Stack pointer

LD SP, src

src: HL, IX, IY

Operation: SP <=src

Flags: S: Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	LD SP, HL	11111001	2
	LD SP, IX or LD SP, IY	11y11101 11111001	4

Notes:

2. y = 0 selects IX and y = 1 selects IY

LDD

Load and Decrement

LDD

Operation: (DE) <= (HL)
BC <= BC - 1
DE <= DE - 1
HL <= HL - 1

Flags: **S:** Unaffected.
Z: Unaffected.
H: Cleared.
P/V: Set if result of decrementing BC is non-zero; cleared otherwise.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
LDD		11101101 10101000	10 (8)

LDDR

Load, Decrement and Repeat

LDDR

Operation: (DE) <= (HL)
BC <= BC - 1
DE <= DE - 1
HL <= HL - 1
repeat if BC != 0

Flags: **S:** Unaffected.
Z: Unaffected
H: Cleared.
P/V: Set if result of decrementing BC is non-zero; cleared otherwise.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	LDDR	<table border="1"><tr><td>11101101</td></tr><tr><td>10111010</td></tr></table>	11101101	10111010	8 + 4i
11101101					
10111010					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each memory read operation.

LDI

Load and Increment

INI

Operation: (DE) <= (HL)
BC <= BC - 1
DE <= DE + 1
HL <= HL + 1

Flags: **S:** Unaffected.
Z: Unaffected.
H: Cleared.
P/V: Set if result of decrementing BC is non-zero; cleared otherwise.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	LDI	<table border="1"><tr><td>11101101</td></tr><tr><td>10100000</td></tr></table>	11101101	10100000	10 (8)
11101101					
10100000					

LDIR

Input, Increment and Repeat

LDIR

Operation: (DE) <= (HL)
BC <= BC - 1
DE <= DE + 1
HL <= HL + 1
repeat if BC != 0

Flags: **S:** Unaffected.
Z: Unaffected.
H: Cleared.
P/V: Set if result of decrementing BC is non-zero; cleared otherwise.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	LDIR	<table border="1"><tr><td>11101101</td></tr><tr><td>10110000</td></tr></table>	11101101	10110000	8 + 4i
11101101					
10110000					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each memory read operation.

MLT

Multiply

MLT src

src: R

Operation: src <= srch * srcl

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	MLT ss	11101110 01ss1100	14 (note 3)

Notes:

1. The **ss** field uses the standard word register encoding.
2. This is an unsigned multiply.
3. A compile-time option exists to change the execution time to 4 clock cycles. This option should only be selected if the technology supports fast carry chains, as it uses a parallel 8x8 multiplier.
4. This instruction is not present in the original Z80, but is a feature of the Z180.

NEG

Negate

NEG

Operation: $A \leftarrow 0 - A$

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic borrow out of bit 3; cleared otherwise.
- P/V:** Set if arithmetic overflow (A was 0x80 before inst); cleared otherwise.
- N:** Cleared.
- C:** Set if arithmetic borrow out of bit 7 (A was not 0x00 before inst); cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	NEG	<table border="1"><tr><td>11101101</td></tr><tr><td>00100100</td></tr></table>	11101101	00100100	4
11101101					
00100100					

NOP

No Operation

NOP

Operation: none

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks	
	NOP	<table border="1"><tr><td>00000000</td></tr></table>	00000000	2
00000000				

OR

Logical OR

OR A, src src: R, IM, IR, X

Operation: $A \leftarrow A \mid \text{src}$

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Cleared.
- P/V:** Set if parity of result is even; cleared otherwise.
- N:** Cleared.
- C:** Cleared.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	OR A, r	10110rrr	2
IM:	OR A, n	11110110 ----n---	4
IR:	OR A, (HL)	10110110	6 (4)
X:	OR A, (IX+d) or OR A, (IY+d)	11y11101 10110110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

OTDM

Output and Decrement Multiple

OTDM

Operation: I/O(0,C) <= (HL)
B <= B - 1
C <= C - 1
HL <= HL - 1

Flags: **S:** Set if the result of decrementing B is negative; cleared otherwise.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Set if there is a borrow out of bit 3 while decrementing B; cleared otherwise.
P/V: Set if the parity of the result of decrementing B is even; cleared otherwise.
N: Set if the byte transferred is negative; cleared otherwise.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OTDM	<table border="1"><tr><td>11101101</td></tr><tr><td>10001011</td></tr></table>	11101101	10001011	10 (8)
11101101					
10001011					

Notes:

1. This instruction is not present in the original Z80, but is a feature of the Z180.

OTDMR

Output, Decrement Multiple and Repeat

OTDMR

Operation: I/O(BC) <= (HL)
B <= B - 1
C <= C - 1
HL <= HL - 1
repeat if B != 0

Flags: **S:** Set if the result of decrementing B is negative; cleared otherwise.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Set if there is a borrow out of bit 3 while decrementing B; cleared otherwise.
P/V: Set if the parity of the result of decrementing B is even; cleared otherwise.
N: Set if the byte transferred is negative; cleared otherwise.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OTDMR	<table border="1"><tr><td>11101101</td></tr><tr><td>10011011</td></tr></table>	11101101	10011011	8 + 4i
11101101					
10011011					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each memory read operation.

OTDR

Output, Decrement and Repeat

OTDR

Operation: I/O(BC) <= (HL)
B <= B - 1
HL <= HL - 1
repeat if B != 0

Flags: **S:** Unaffected.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Unaffected.
P/V: Unaffected.
N: Set.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OTDR	<table border="1"><tr><td>11101101</td></tr><tr><td>10111011</td></tr></table>	11101101	10111011	8 + 4i
11101101					
10111011					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each memory read operation.
3. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

OTIM

Output and Increment Multiple

OTIM

Operation: I/O(BC) \leq (HL)
B \leq B - 1
C \leq C + 1
HL \leq HL + 1

Flags: **S:** Set if the result of decrementing B is negative; cleared otherwise.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Set if there is a borrow out of bit 3 while decrementing B; cleared otherwise.
P/V: Set if the parity of the result of decrementing B is even; cleared otherwise.
N: Set if the byte transferred is negative; cleared otherwise.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OTIM	<table border="1"><tr><td>11101101</td></tr><tr><td>10000011</td></tr></table>	11101101	10000011	10 (8)
11101101					
10000011					

Notes:

1. This instruction is not present in the original Z80, but is a feature of the Z180.

OTIMR

Output, Increment Multiple and Repeat

OTIMR

Operation: I/O(BC) <= (HL)
B <= B - 1
C <= C + 1
HL <= HL + 1
repeat if B != 0

Flags: **S:** Set if the result of decrementing B is negative; cleared otherwise.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Set if there is a borrow out of bit 3 while decrementing B; cleared otherwise.
P/V: Set if the parity of the result of decrementing B is even; cleared otherwise.
N: Set if the byte transferred is negative; cleared otherwise.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OTIMR	<table border="1"><tr><td>11101101</td></tr><tr><td>10010011</td></tr></table>	11101101	10010011	8 + 4i
11101101					
10010011					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each memory read operation.
3. This instruction is not present in the original Z80, but is a feature of the Z180.

OTIR

Output, Increment and Repeat

OTIR

Operation: I/O(BC) <= (HL)
B <= B - 1
HL <= HL + 1
repeat if B != 0

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Set if result of decrementing B is zero; cleared otherwise.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OTIR	<table border="1"><tr><td>11101101</td></tr><tr><td>10110011</td></tr></table>	11101101	10110011	8 + 4i
11101101					
10110011					

Notes:

1. This instruction can be interrupted after each iteration. The address saved on the stack in this case is the address of this instruction, allowing completion of the instruction after the interrupt service routine
2. Interrupts are sampled during each memory read operation.
3. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

OUT

Output

OUT dst, A

dst: DA

Operation: I/O(A:n) <= A

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OUT (n), A	<table border="1"><tr><td>11010011</td></tr><tr><td>----n---</td></tr></table>	11010011	----n---	8 (6)
11010011					
----n---					

OUT

Output

OUT (C), r

src: R

Operation: I/O(BC) <= r

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OUT (C), r	<table border="1"><tr><td>11101101</td></tr><tr><td>01rrr001</td></tr></table>	11101101	01rrr001	8 (6)
11101101					
01rrr001					

Notes:

1. The **rrr** field uses the standard register select encoding

OUT0

Output (page 0)

OUT0 (n), r src: R

Operation: I/O(0,n) <= r

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks			
	OUT0 (n), r	<table border="1"><tr><td>11101101</td></tr><tr><td>00rrr001</td></tr><tr><td>----n---</td></tr></table>	11101101	00rrr001	----n---	10 (8)
11101101						
00rrr001						
----n---						

Notes:

1. The **rrr** field uses the standard register select encoding
2. This instruction is not present in the original Z80, but is a feature of the Z180.

OUTD

Output and Decrement

OUTD

Operation: I/O(BC) <= (HL)
B <= B - 1
HL <= HL -1

Flags: **S:** Unaffected.
Z: Set if result of decrementing B is zero; cleared otherwise.
H: Unaffected.
P/V: Unaffected.
N: Set.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OUTD	<table border="1"><tr><td>11101101</td></tr><tr><td>10101011</td></tr></table>	11101101	10101011	10 (8)
11101101					
10101011					

Notes:

1. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

OUTI

Output and Increment

OUTI

Operation: I/O(BC) <= (HL)
B <= B - 1
HL <= HL + 1

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Set if result of decrementing B is zero; cleared otherwise.
N: Cleared.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	OUTI	<table border="1"><tr><td>11101101</td></tr><tr><td>10100011</td></tr></table>	11101101	10100011	10 (8)
11101101					
10100011					

Notes:

1. For the original Z80, the **S**, **H** and **P/V** flags are undefined.

PUSH

Push to Stack

PUSH src

src: RR, IX, IY

Operation: (SP-1) <= src[msb]
(SP-2) <= src[lsb]
SP <= SP - 2

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
RR:	PUSH tt	11tt0101	8 (6)
IX, IY	PUSH IX or PUSH IY	11y11101	10 (8)
		11100101	

Notes:

1. The **tt** field uses the standard word register encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY

RES

Bit Reset

RES b, dst

src: R, IR, X

Operation: dst[b] <= 0

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	RES b, r	<table border="1"> <tr><td>11001011</td></tr> <tr><td>10bbbrrr</td></tr> </table>	11001011	10bbbrrr	4		
11001011							
10bbbrrr							
IR:	RES b, (HL)	<table border="1"> <tr><td>10100110</td></tr> <tr><td>10bbb1110</td></tr> </table>	10100110	10bbb1110	10 (8)		
10100110							
10bbb1110							
X:	RES b, (IX+d) or RES (IY+d)	<table border="1"> <tr><td>11y11101</td></tr> <tr><td>11001011</td></tr> <tr><td>----d---</td></tr> <tr><td>10bbb110</td></tr> </table>	11y11101	11001011	----d---	10bbb110	12 (10)
11y11101							
11001011							
----d---							
10bbb110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.
3. The **bbb** field uses normal binary encoding.

RET

Return from Subroutine

RET

Operation: PC[lsb] <= (SP)
 PC[msb] <= (SP+1)
 SP <= SP + 2

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	RET	11001001	10

RET

Conditional Return from Subroutine

RET cc

Operation: if (cc = true) begin
 PC[lsb] <= (SP)
 PC[msb] <= (SP+1)
 SP <= SP + 2
end

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks	
	RET cc	<table border="1"><tr><td>11fff000</td></tr></table>	11fff000	10 (taken) 2 (not taken)
11fff000				

Notes:

1.	Mnemonic	Encoding (fff)	Meaning	Flag case
	NZ	000	Non-zero	Z = 0
	Z	001	Zero	Z = 1
	NC	010	Non-carry	C = 0
	C	011	Carry	C = 1
	PO	100	Parity Odd	P/V = 0
	PE	101	Parity Even	P/V = 1
	P	110	Plus	S = 0
	M	111	Minus	S = 1

RETI

Return from Interrupt

RETI

Operation: PC[lsb] <= (SP)
 PC[msb] <= (SP+1)
 SP <= SP + 2

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	RETI	<table border="1"><tr><td>11101101</td></tr><tr><td>01001101</td></tr></table>	11101101	01001101	12
11101101					
01001101					

Notes:

1. This instruction activates the dedicated RETI signal out of the core.

RETN

Return from Non-Maskable Interrupt

RETN

Operation: PC[lsb] <= (SP)
PC[msb] <= (SP+1)
SP <= SP + 2
IFF2 <= IFF1

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	RETN	11001001 01000101	12

RL

Rotate Left

RL src

src: R, IR, X

Operation: {CF, src} <= {src, CF}

Flags: **S:** Set if result is negative; cleared otherwise.
Z: Set if result is zero; cleared otherwise.
H: Cleared.
P/V: Set if parity of result is even; cleared otherwise.
N: Cleared.
C: Data from bit 7.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	RL r	<table border="1"><tr><td>11001011</td></tr><tr><td>00010rrr</td></tr></table>	11001011	00010rrr	4		
11001011							
00010rrr							
IR:	RL (HL)	<table border="1"><tr><td>10100110</td></tr><tr><td>00010110</td></tr></table>	10100110	00010110	10 (8)		
10100110							
00010110							
X:	RL (IX+d) or RL (IY+d)	<table border="1"><tr><td>11y11101</td></tr><tr><td>11001011</td></tr><tr><td>----d---</td></tr><tr><td>00010110</td></tr></table>	11y11101	11001011	----d---	00010110	12 (10)
11y11101							
11001011							
----d---							
00010110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.

RLA

Rotate Left Accumulator

RLA

Operation: {CF, A} <= {A, CF}

Flags: **S:** Unaffected
 Z: Unaffected.
 H: Cleared.
 P/V: Unaffected.
 N: Cleared.
 C: Data from bit 7.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	RLA	00010111	2

RLCA

Rotate Left Circular Accumulator

RLCA

Operation: {CF, A} <= {A, A[7]}

Flags: **S:** Unaffected
 Z: Unaffected.
 H: Cleared.
 P/V: Unaffected.
 N: Cleared.
 C: Data from bit 7.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	RLCA	0000111	2

RLD

Rotate Left Digit

RLD

Operation: {A, (HL)} <= {A[7:4], (HL), A[3:0]}

Flags: **S:** Set if A is negative after the operation; cleared otherwise.
Z: Set if A is zero after the operation; cleared otherwise.
H: Cleared.
P/V: Set if parity of A is even after the operation; cleared otherwise.
N: Cleared.
C: Unaffected

Addressing Modes	Assembly Syntax	Encoding	Clocks
RLD		11101101 01101111	10 (8)

RR

Rotate Right

RR src

src: R, IR, X

Operation: {src, CF} <= {CF, src}

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Cleared.
- P/V:** Set if parity of result is even; cleared otherwise.
- N:** Cleared.
- C:** Data from bit 0.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	RR r	<table border="1"><tr><td>11001011</td></tr><tr><td>00011rrr</td></tr></table>	11001011	00011rrr	4		
11001011							
00011rrr							
IR:	RR (HL)	<table border="1"><tr><td>10100110</td></tr><tr><td>00011110</td></tr></table>	10100110	00011110	10 (8)		
10100110							
00011110							
X:	RR (IX+d) or RR (IY+d)	<table border="1"><tr><td>11y11101</td></tr><tr><td>11001011</td></tr><tr><td>----d---</td></tr><tr><td>00011110</td></tr></table>	11y11101	11001011	----d---	00011110	12 (10)
11y11101							
11001011							
----d---							
00011110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.

RRA

Rotate Right Accumulator

RRA

Operation: {A, CF} <= {CF, A}

Flags: **S:** Unaffected
 Z: Unaffected.
 H: Cleared.
 P/V: Unaffected.
 N: Cleared.
 C: Data from bit 0.

Addressing Modes	Assembly Syntax	Encoding	Clocks	
	RRA	<table border="1"><tr><td>00011111</td></tr></table>	00011111	2
00011111				

RRC

Rotate Right Circular

RRC src

src: R, IR, X

Operation: {src, CF} <= {src[0], src}

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Cleared.
- P/V:** Set if parity of result is even; cleared otherwise.
- N:** Cleared.
- C:** Data from bit 0.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	RRC r	<table border="1"><tr><td>11001011</td></tr><tr><td>00001rrr</td></tr></table>	11001011	00001rrr	4		
11001011							
00001rrr							
IR:	RRC (HL)	<table border="1"><tr><td>10100110</td></tr><tr><td>00001110</td></tr></table>	10100110	00001110	10 (8)		
10100110							
00001110							
X:	RRC (IX+d) or RRC (IY+d)	<table border="1"><tr><td>11y11101</td></tr><tr><td>11001011</td></tr><tr><td>----d---</td></tr><tr><td>00001110</td></tr></table>	11y11101	11001011	----d---	00001110	12 (10)
11y11101							
11001011							
----d---							
00001110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.

RRCA

Rotate Right Circular Accumulator

RRCA

Operation: {A, CF} <= {A[0], A}

Flags: **S:** Unaffected
 Z: Unaffected.
 H: Cleared.
 P/V: Unaffected.
 N: Cleared.
 C: Data from bit 0.

Addressing Modes	Assembly Syntax	Encoding	Clocks	
	RRCA	<table border="1"><tr><td>00001111</td></tr></table>	00001111	2
00001111				

RRD

Rotate Right Digit

RRD

Operation: {A, (HL)} <= {A[7:4], (HL)[3:0], A[3:0], (HL)[7:4]}

Flags:

- S:** Set if A is negative after the operation; cleared otherwise.
- Z:** Set if A is zero after the operation; cleared otherwise.
- H:** Cleared.
- P/V:** Set if parity of A is even after the operation; cleared otherwise.
- N:** Cleared.
- C:** Unaffected

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	RRD	<table border="1"><tr><td>11101101</td></tr><tr><td>01100111</td></tr></table>	11101101	01100111	10 (8)
11101101					
01100111					

RST

Restart

RST v

Operation: $SP \leq SP - 2$
 $(SP) \leq PC$
 $PC \leq v$

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks	
	RST v	<table border="1"><tr><td>11vvv111</td></tr></table>	11vvv111	8
11vvv111				

Notes:

1.	Mnemonic	Encoding (vvv)	Restart Address
	0	000	0x0000
	0x8	001	0x0008
	0x10	010	0x0010
	0x18	011	0x0018
	0x20	100	0x0020
	0x28	101	0x0028
	0x30	110	0x0030
	0x38	111	0x0038

SBC

Subtract With Carry

SBC A, src

src: R, IM, IR, X

Operation: $A \leftarrow A - \text{src} - \text{CF}$

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic borrow out of bit 3; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Cleared.
- C:** Set if arithmetic borrow out of bit 7; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	SBC A, r	1001rrr	2
IM:	SBC A, n	11011110 ----n---	4
IR:	SBC A, (HL)	10011110	6 (4)
X:	SBC A, (IX+d) or SBC A, (IY+d)	11y11101 10011110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

SBC

Subtract With Carry (Word)

SBC HL, src

src: RR

Operation: HL \leftarrow HL - src - CF

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic borrow out of bit 11; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Cleared.
- C:** Set if arithmetic carry out of bit 15; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
RR:	SBC HL, ss	<table border="1"><tr><td>11101101</td></tr><tr><td>01ss0010</td></tr></table>	11101101	01ss0010	4
11101101					
01ss0010					

Notes:

1. The **ss** field uses the standard word register encoding.

SCF

Set Carry Flag

CCF

Operation: CF \leftarrow 1

Flags: S: Unaffected.
Z: Unaffected.
H: Cleared.
P/V: Unaffected.
N: Cleared.
C: Set.

Addressing Modes	Assembly Syntax	Encoding	Clocks
	SCF	00110111	2

SET

Bit Set

SET b, dst

src: R, IR, X

Operation: dst[b] <= 1

Flags: **S:** Unaffected.
Z: Unaffected.
H: Unaffected.
P/V: Unaffected.
N: Unaffected.
C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	SET b, r	<table border="1"><tr><td>11001011</td></tr><tr><td>11bbbrrr</td></tr></table>	11001011	11bbbrrr	4		
11001011							
11bbbrrr							
IR:	SET b, (HL)	<table border="1"><tr><td>10100110</td></tr><tr><td>11bbb1110</td></tr></table>	10100110	11bbb1110	10 (8)		
10100110							
11bbb1110							
X:	SET b, (IX+d) or SET b, (IY+d)	<table border="1"><tr><td>11y11101</td></tr><tr><td>11001011</td></tr><tr><td>----d---</td></tr><tr><td>11bbb110</td></tr></table>	11y11101	11001011	----d---	11bbb110	12 (10)
11y11101							
11001011							
----d---							
11bbb110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.
3. The **bbb** field uses normal binary encoding.

SLA

Shift Left Arithmetic

SLA src src: R, IR, X

Operation: {CF, src} <= {src, 0}

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Cleared.
- P/V:** Set if parity of result is even; cleared otherwise.
- N:** Cleared.
- C:** Data from bit 7.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	SLA r	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px;">11001011</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">00100rrr</td></tr> </table>	11001011	00100rrr	4		
11001011							
00100rrr							
IR:	SLA (HL)	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px;">10100110</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">00100110</td></tr> </table>	10100110	00100110	10 (8)		
10100110							
00100110							
X:	SLA (IX+d) or SLA (IY+d)	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px;">11y11101</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">11001011</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">----d---</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">00100110</td></tr> </table>	11y11101	11001011	----d---	00100110	12 (10)
11y11101							
11001011							
----d---							
00100110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.

SLP

Sleep

SLP

Operation: activate SLEEP signal and wait for interrupt

Flags: **S:** Unaffected.
 Z: Unaffected.
 H: Unaffected.
 P/V: Unaffected.
 N: Unaffected.
 C: Unaffected.

Addressing Modes	Assembly Syntax	Encoding	Clocks		
	SLP	<table border="1"><tr><td>11101101</td></tr><tr><td>01110110</td></tr></table>	11101101	01110110	6 + 2n
11101101					
01110110					

Notes:

1. The CPU halts, with an idle bus, until an interrupt is requested. During Sleep the **mem_addr_bus** and **io_addr_bus** are driven with 0x0000, and the **mem_data_bus** and **io_data_bus** are driven with 0x00.
2. In the case of an NMI or enabled maskable interrupt the address pushed to the stack during the interrupt acknowledge is the address of the next instruction.
3. If interrupts are disabled a maskable interrupt request during Sleep causes the CPU to resume execution with the next instruction. This saves time when restarting from Sleep.
4. This instruction is not present in the original Z80, but is a feature of the Z180.

SRA

Shift Right Arithmetic

SRA src src: R, IR, X

Operation: {src, CF} <= {src[7], src}

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Cleared.
- P/V:** Set if parity of result is even; cleared otherwise.
- N:** Cleared.
- C:** Data from bit 0.

Addressing Modes	Assembly Syntax	Encoding	Clocks				
R:	SRA r	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px;">11001011</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">00101rrr</td></tr> </table>	11001011	00101rrr	4		
11001011							
00101rrr							
IR:	SRA (HL)	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px;">10100110</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">00101110</td></tr> </table>	10100110	00101110	10 (8)		
10100110							
00101110							
X:	SRA (IX+d) or SRA (IY+d)	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px;">11y11101</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">11001011</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">----d---</td></tr> <tr><td style="border: 1px solid black; padding: 2px;">00101110</td></tr> </table>	11y11101	11001011	----d---	00101110	12 (10)
11y11101							
11001011							
----d---							
00101110							

Notes:

1. The **rrr** field uses the standard register select encoding.
2. **y** = 0 selects IX and **y** = 1 selects IY.

SUB

Subtract

SUB A, src src: R, IM, IR, X

Operation: $A \leftarrow A - \text{src}$

Flags:

- S:** Set if result is negative; cleared otherwise.
- Z:** Set if result is zero; cleared otherwise.
- H:** Set if arithmetic borrow out of bit 3; cleared otherwise.
- P/V:** Set if arithmetic overflow; cleared otherwise.
- N:** Cleared.
- C:** Set if arithmetic borrow out of bit 7; cleared otherwise.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	SUB A, r	10010rrr	2
IM:	SUB A, n	11010110 ----n---	4
IR:	SUB A, (HL)	10010110	6 (4)
X:	SUB A, (IX+d) or SUB A, (IY+d)	11y11101 10010110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

TST

Test

TST src src: R, IM, IR

Operation: A & src

Flags: **S:** Set if result is negative; cleared otherwise.
Z: Set if result is zero; cleared otherwise.
H: Set.
P/V: Set if parity of result is even; cleared otherwise.
N: Cleared.
C: Cleared.

Addressing Modes	Assembly Syntax	Encoding	Clocks			
R:	TST r	<table border="1"><tr><td>11101101</td></tr><tr><td>00rrr100</td></tr></table>	11101101	00rrr100	4	
11101101						
00rrr100						
IM:	TST n	<table border="1"><tr><td>11101101</td></tr><tr><td>01100100</td></tr><tr><td>----n---</td></tr></table>	11101101	01100100	----n---	6
11101101						
01100100						
----n---						
IR:	TST (HL)	<table border="1"><tr><td>11101101</td></tr><tr><td>00110100</td></tr></table>	11101101	00110100	8 (6)	
11101101						
00110100						

Notes:

1. The **rrr** field uses the standard register select encoding
2. This instruction is not present in the original Z80, but is a feature of the Z180.

TSTIO

Test I/O

TSTIO n

Operation: I/O(0,C) & n

Flags: **S:** Set if result is negative; cleared otherwise.
Z: Set if result is zero; cleared otherwise.
H: Set.
P/V: Set if parity of result is even; cleared otherwise.
N: Cleared.
C: Cleared.

Addressing Modes	Assembly Syntax	Encoding	Clocks			
	TSTIO n	<table border="1"><tr><td>11101101</td></tr><tr><td>01110100</td></tr><tr><td>----n---</td></tr></table>	11101101	01110100	----n---	10
11101101						
01110100						
----n---						

Notes:

1. This instruction is not present in the original Z80, but is a feature of the Z180.

XOR

Logical Exclusive-OR

XOR A, src

src: R, IM, IR, X

Operation: $A \leftarrow A \wedge \text{src}$

Flags: **S:** Set if result is negative; cleared otherwise.
Z: Set if result is zero; cleared otherwise.
H: Cleared.
P/V: Set if parity of result is even; cleared otherwise.
N: Cleared.
C: Cleared.

Addressing Modes	Assembly Syntax	Encoding	Clocks
R:	XOR A, r	10101rrr	2
IM:	XOR A, n	11101110 ----n---	4
IR:	XOR A, (HL)	10101110	6 (4)
X:	XOR A, (IX+d) or XOR A, (IY+d)	11y11101 10101110 ----d---	10 (8)

Notes:

1. The **rrr** field uses the standard register select encoding
2. **y** = 0 selects IX and **y** = 1 selects IY

Memory Management Unit

The Memory Management Unit (MMU) expands the logical address space of 64K bytes (16-bit address) to a physical address space of 1M bytes (20-bit address), using variable-sized segments.

Features:

- Three memory segments, aligned on 4k boundaries
- Two eight-bit base registers, one implied base register of 0x00

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
MMU Common Base Register	CBR	0x0038	R/W	00000000
MMU Bank Base Register	BBR	0x0039	R/W	00000000
MMU Common/Bank Area Register	CBAR	0x003A	R/W	11110000

Register Descriptions

Common Base Register (CBR) (Address = 0x0038)		
Bit(s)	Value	Description
7:0		Eight bits of physical address offset to use if: CBAR[7:4] <= Addr[15:12]

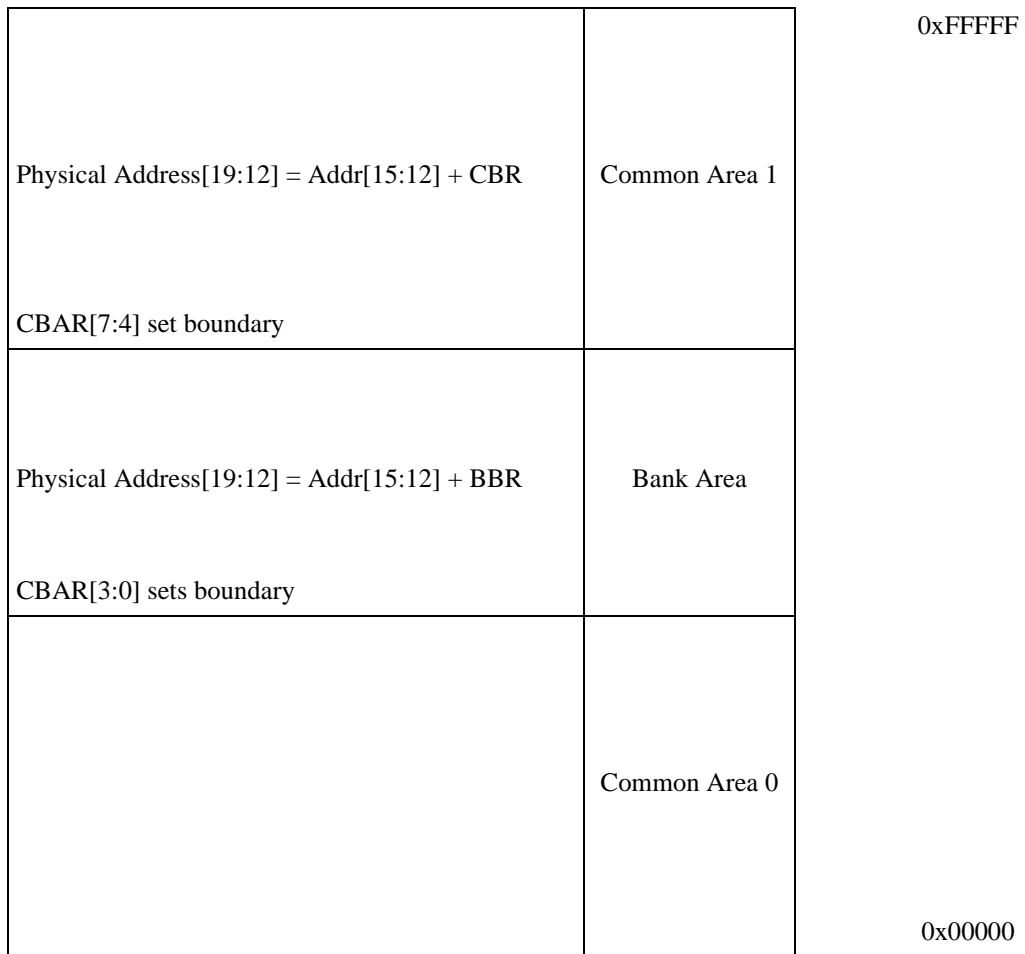
Bank Base Register (BBR) (Address = 0x0039)		
Bit(s)	Value	Description
7:0		Eight bits of physical address offset to use if: CBAR[3:0] <= Addr[15:12] < CBAR[7:4]

MMU Common/Bank Area Register (CBAR) (Address = 0x003A)		
Bit(s)	Value	Description
7:4	Write	Boundary value for switching from BBR to CBR for translation.
3:0	Write	Boundary value for switching from none to BBR for translation.

MMU Operation Details

The Memory Management Unit divides the 64K logical address space into three variable-sized segments, with each segment a multiple of 4K bytes. Depending on the MMU programming one, two, or all three segments may be used at once.

Only two of the segments can be relocated, using a base register in the MMU. The third segment always starts at logical address 0x0000. The figure below shows the typical case of three segments:



Interrupt Control

The Interrupt Control (ICTL) manages all of the on-chip and external interrupt requests. It automatically prioritizes interrupt requests and generates interrupt vectors where appropriate.

Features:

- Fixed prioritization of all interrupt requests.
- Relocatable interrupt vectors for all on-chip peripherals.
- Individual enables for external interrupt requests (**int0_req**, **int1_req** and **int2_req**).
- Automatic selection of Interrupt Mode 2 for internal and two external interrupt requests.

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
Interrupt Vector Low Register	IL	0x0033	R/W	00000000
Interrupt/Trap Control Register	ITC	0x0034	R/W	00111001

Register Descriptions

Interrupt Vector Low Register		(IL)	(Address = 0x0033)
Bit(s)	Value	Description	
7:5		<p>These are the three most-significant bits of the interrupt vector returned during an interrupt acknowledge transaction for the on-chip interrupts, as well as for the int1_req and int2_req inputs. The other bits of the vector are:</p> <pre> int1_req: 00000 int2_req: 00010 prt0: 00100 prt1: 00110 dma0: 01000 dma1: 01010 csio: 01100 asci0: 01110 asci1: 10000 </pre> <p>Note that int0_req does not automatically use Interrupt Mode 2, and if a vector is required for this interrupt request it must be externally supplied.</p>	
4:0		These bits are reserved and should always be written with zeros.	

Interrupt/Trap Control Register		(ITC)	(Address = 0x0034)
Bit(s)	Value	Description	
7	0	No illegal instruction fetch since last write of zero to this bit.	
	1	An illegal instruction has been fetched. This status is latched until this bit is written with zero.	
6	0	Last illegal instruction fetch was a two-byte opcode.	
	1	Last illegal instruction fetch was a three-byte opcode.	
5:3		These bits are reserved. Reads always return ones.	
2	0	Disable int2_req interrupt request.	
	1	Enable int2_req interrupt request.	
1	0	Disable int1_req interrupt request.	
	1	Enable int1_req interrupt request.	
0	0	Disable int0_req interrupt request.	
	1	Enable int0_req interrupt request.	

Direct Memory Access

Two Direct memory Access (DMA) channels provide memory-to-memory or memory-to-I/O data transfers in parallel with CPU operation. The two channels are not identical, but have similar capabilities.

Features:

- 20-bit physical memory address, 16-bit I/O address
- Channel 0 supports memory-memory and memory-I/O transfers
- Channel 1 supports only memory-I/O transfers
- Memory address increment, decrement, or fixed
- I/O address always fixed
- Maximum transfer length of 65,536 bytes
- **dreqb** (external request) input for each channel, with programmable polarity
- **tendb** (Transfer End or DMA Complete) output for each channel
- Mode to toggle between channels, allows simulating scatter/gather operation

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
DMA Source Address Register 0 Low	SAR0L	0x0020	R/W	00000000
DMA Source Address Register 0 High	SAR0H	0x0021	R/W	00000000
DMA Source Address Register 0 Page	SAR0P	0x0022	R/W	00000000
DMA Destination Address Register 0 Low	DAR0L	0x0023	R/W	00000000
DMA Destination Address Register 0 High	DAR0H	0x0024	R/W	00000000
DMA Destination Address Register 0 Page	DAR0P	0x0025	R/W	00000000
DMA Byte Count Register 0 Low	BCR0L	0x0026	R/W	00000000
DMA Byte Count Register 0 High	BCR0H	0x0027	R/W	00000000
DMA Memory Address Register 1 Low	MAR1L	0x0028	R/W	00000000
DMA Memory Address Register 1 High	MAR1H	0x0029	R/W	00000000
DMA Memory Address Register 1 Page	MAR1P	0x002A	R/W	00000000
DMA I/O Address Register 1 Low	IAR1L	0x002B	R/W	00000000
DMA I/O Address Register 1 High	IAR1H	0x002C	R/W	00000000
DMA I/O Address Register 1 Page	IAR1P	0x002D	R/W	00000000
DMA Byte Count Register 1 Low	BCR1L	0x002E	R/W	00000000
DMA Byte Count Register 1 High	BCR1H	0x002F	R/W	00000000
DMA Status Register	DSTAT	0x0030	R/W	00000000
DMA Mode Register	DMODE	0x0031	R/W	00000000
DMA/Wait Control Register	DCNTL	0x0032	R/W	11110000

Register Descriptions

DMA Source Address Register 0		(SAR0L)	(Address = 0x0020)
		(SAR0H)	(Address = 0x0021)
		(SAR0P)	(Address = 0x0022)
Bit(s)	Value	Description	
23:20		These bits are reserved and will always be read as zero.	
19:16		DMA 0 source address page. Not used if the DMA 0 source is I/O.	
15:8		DMA 0 source address MSB.	
7:0		DMA 0 source address LSB.	

DMA Destination Address Register 0		(DAR0L)	(Address = 0x0023)
		(DAR0H)	(Address = 0x0024)
		(DAR0P)	(Address = 0x0025)
Bit(s)	Value	Description	
23:20		These bits are reserved and will always be read as zero.	
19:16		DMA 0 destination address page. Not used if the DMA 0 destination is I/O.	
15:8		DMA 0 destination address MSB.	
7:0		DMA 0 destination address LSB.	

DMA Byte Count Register 0		(BCR0L)	(Address = 0x0026)
		(BCR0H)	(Address = 0x0027)
Bit(s)	Value	Description	
15:8		DMA 0 byte count MSB.	
7:0		DMA 0 byte count LSB.	

DMA Memory Address Register 1		(MAR1L)	(Address = 0x0028)
		(MAR1H)	(Address = 0x0029)
		(MAR1P)	(Address = 0x002A)
Bit(s)	Value	Description	
23:20		These bits are reserved and will always be read as zero.	
19:16		DMA 1 memory address page.	
15:8		DMA 1 memory address MSB.	
7:0		DMA 1 memory address LSB.	

DMA I/O Address Register 1		(IAR1L)	(Address = 0x002B)
		(IAR1H)	(Address = 0x002C)
		(IAR1P)	(Address = 0x002D)
Bit(s)	Value	Description	
23	0	DMA channels are independent.	
	1	Toggle between channels at terminal count. Automatically switches selected internal request between channels.	
22	0	DMA 0 is the currently selected channel. This bit is valid only when the channel toggle function is enabled in this register.	
	1	DMA 1 is the currently selected channel. This bit is valid only when the channel toggle function is selected in this register.	
21:19		These bits are reserved, but are R/W.	
18:16	000	DMA 1 is controlled by dreqlb .	
	001	DMA 1 connected to ASCI0 (receive if DMA 1 source is I/O, transmit if DMA 1 destination is I/O).	
	010	DMA 1 connected to ASCI1 (receive if DMA 1 source is I/O, transmit if DMA 1 destination is I/O).	
	011	DMA 1 is controlled by dreql0b .	
	1xx	These bit combinations are reserved and should not be used.	
15:8		DMA 1 I/O address MSB.	
7:0		DMA 1 I/O address LSB.	

DMA Byte Count Register 1		(BCR1L)	(Address = 0x002E)
		(BCR1H)	(Address = 0x002F)
Bit(s)	Value	Description	
15:8		DMA 1 byte count MSB.	
7:0		DMA 1 byte count LSB.	

DMA Status Register		(DSTAT)	(Address = 0x0030)
Bit(s)	Value	Description	
7	0	Disable DMA 1. An interrupt will be requested while this bit is cleared and DMA 1 interrupts are enabled.	
	1	Enable DMA 1. Note that this bit can only be updated if bit 5 is cleared during the write.	
6	0	Disable DMA 0. An interrupt will be requested while this bit is cleared and DMA 0 interrupts are enabled.	
	1	Enable DMA 0. Note that this bit can only be updated if bit 4 is cleared during the write.	
5 (wr-only)	0	Allow bit 7 of this register to be updated with this write.	
	1	Do not allow bit 7 of this register to be updated with this write. This bit always returns one when read.	
4 (wr-only)	0	Allow bit 6 of this register to be updated with this write.	
	1	Do not allow bit 6 of this register to be updated with this write. This bit always returns one when read.	
3	0	Disable DMA 1 interrupt.	
	1	Enable DMA 1 interrupt.	
2	0	Disable DMA 0 interrupt.	
	1	Enable DMA 0 interrupt.	
1		This bit is reserved and should always be written with zero.	
0 (rd-only)	0	DMA is disabled. This bit is cleared by a Non-maskable Interrupt request (nmi_req) to inhibit DMA operation during the NMI service routine.	
	1	DMA is enabled. This bit is not set directly, but will be set whenever either bit 7 or bit 6 of this register are written with 1.	

DMA Mode Register (DMODE) (Address = 0x0031)		
Bit(s)	Value	Description (Async mode only)
7:6		These bits are reserved
5:4	00	DMA 0 destination is memory. Auto-increment destination address after each transfer.
	01	DMA 0 destination is memory. Auto-decrement destination address after each transfer.
	10	DMA 0 destination is memory, with a fixed address.
	11	DMA 0 destination is I/O, with a fixed address.
3:2	00	DMA 0 source is memory. Auto-increment source address after each transfer.
	01	DMA 0 source is memory. Auto-decrement source address after each transfer.
	10	DMA 0 source is memory, with a fixed address.
	11	DMA 0 source is I/O, with a fixed address. Note that the Y90 allows I/O-to-I/O DMA transfers.
1	0	DMA 0 memory-to-memory transfers are one byte at a time.
	1	DMA 0 memory-to-memory transfers burst for the entire byte count.
0		This bit is reserved and always returns zero when read.

DMA/Wait Control Register (DCNTL) (Address = 0x00312)		
Bit(s)	Value	Description (Async mode only)
7:6	00	Memory transfers have 0 wait states.
	01	Memory transfers have 1 wait state.
	10	Memory transfers have 2 wait states.
	11	Memory transfers have 3 wait states.
5:4	00	I/O transfers have 0 wait states.
	01	I/O transfers have 1 wait state.
	10	I/O transfers have 2 wait states.
	11	I/O transfers have 3 wait states.
3	0	dreq1b is level-sensitive. DMA transfers continue as long as dreq1b is active.
	1	dreq1b is edge-sensitive. One DMA transfer per falling edge on dreq1b .
2	0	dreq0b is level-sensitive. DMA transfers continue as long as dreq0b is active.
	1	dreq0b is edge-sensitive. One DMA transfer per falling edge on dreq0b .
1:0	00	DMA 0 transfers are memory-to-I/O. Memory address auto-increments and I/O address is fixed.
	01	DMA 0 transfers are memory-to-I/O. Memory address auto-decrements and I/O address is fixed.
	10	DMA 0 transfers are I/O-to-memory. Memory address auto-increments and I/O address is fixed.
	11	DMA 0 transfers are I/O-to-memory. Memory address auto-decrements and I/O address is fixed.

Async Serial

Each of the two Async Serial Communications Interface (ASCI) channels provide asynchronous communications capabilities.

Features:

- Full-duplex, 7 or 8 bits/character
- Optional odd or even parity
- Optional multiprocessor bit
- One or two Stop bits
- One byte of buffering in both receiver and transmitter
- Automatic operation with on-chip DMA
- Overrun Error, Framing Error and Parity Error detection
- 16x or 64x sampling clock (does not support 1x sampling clock of Z8L180)
- **ctsb**, **dcdb** and **rtsb** modem control signals for both channels
- Detect and send Break
- External clock input, dedicated divider, or programmable 16-bit divider for baud-rate

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
ASCII Control Register A, Channel 0	CNTLA0	0x0000	R/W	0001x000
ASCII Control Register A, Channel 1	CNTLA1	0x0001	R/W	0001x000
ASCII Control Register B, Channel 0	CNTLB0	0x0002	R/W	x0x00111
ASCII Control Register B, Channel 1	CNTLB1	0x0003	R/W	x0x00111
ASCII Status Register, Channel 0	STAT0	0x0004	R/W	00000xx0
ASCII Status Register, Channel 1	STAT1	0x0005	R/W	00000xx0
ASCII Transmit Data Register, Channel 0	TDR0	0x0006	R/W	xxxxxxxx
ASCII Transmit Data Register, Channel 1	TDR1	0x0007	R/W	xxxxxxxx
ASCII Receive Data Register, Channel 0	RDR0	0x0008	R/W	xxxxxxxx
ASCII Receive Data Register, Channel 1	RDR1	0x0009	R/W	xxxxxxxx
ASCII Extension Control Register, Channel 0	ECNTL0	0x0012	R/W	00000000
ASCII Extension Control Register, Channel 1	ECNTL1	0x0013	R/W	00000000
ASCII Time Constant Low Register, Channel 0	TCL0	0x001A	R/W	00000000
ASCII Time Constant High Register, Channel 0	TCH0	0x001B	R/W	00000000
ASCII Time Constant Low Register, Channel 1	TCL1	0x001C	R/W	00000000
ASCII Time Constant High Register, Channel 1	TCH1	0x001D	R/W	00000000

Register Descriptions

ASCI Control Register A (CNTLA0) (Address = 0x0000)		
CNTLA1 (Address = 0x0001)		
Bit(s)	Value	Description
7	0	Do not filter receive bytes based on the MP bit.
	1	If multiprocessor mode is enabled, receive only bytes with the MP bit set to one.
6	0	Disable the receiver. This selection is forced by IOSTOP mode.
	1	Enable the receiver.
5	0	Disable the transmitter. This selection is forced by IOSTOP mode.
	1	Enable the transmitter.
4	0	Drive the corresponding rtsb output Low.
	1	Drive the corresponding rtsb output High.
3 (read)	0	Either multiprocessor mode is disabled, or the last received MP bit was set to zero.
	1	Multiprocessor mode is enabled and the last received MP bit was set to one.
3 (write)	0	Clear all of the error flags (Overrun, Framing Error and Parity Error).
	1	No effect.
2	0	Select 7 bits per character.
	1	Select 8 bits per character.
1	0	Disable parity generation and checking.
	1	Enable parity generation and checking.
0	0	Select 1 stop bit.
	1	Select 2 stop bits. Note that the receiver always checks for one stop bit.

ASCI Control Register B		(CNTLB0)	(Address = 0x0002)
		(CNTLB1)	(Address = 0x0003)
Bit(s)	Value	Description	
7	0	Do not filter receive bytes based on the MP bit.	
	1	In multiprocessor mode, receive only bytes with the MP bit set to one.	
6	0	Disable multiprocessor mode.	
	1	Enable multiprocessor mode. In multiprocessor mode there is no parity, but the parity bit location is used to tag the data as either data (MP bit is zero) or an address (MP bit is one).	
5 (read)	0	The corresponding ctsb input is Low.	
	1	The corresponding ctsb input is High.	
5 (write)	0	System clock divided by 10 drives the internal baud rate generation.	
	1	System clock divided by 30 drives the internal baud rate generation.	
4	0	Select even parity.	
	1	Select odd parity.	
3	0	Select a sampling clock rate of 16x.	
	1	Select a sampling clock rate of 64x.	
2:0	000	Select internal clock, used directly.	
	001	Select internal clock divided by 2.	
	010	Select internal clock divided by 4.	
	011	Select internal clock divided by 8.	
	100	Select internal clock divided by 16.	
	101	Select internal clock divided by 32.	
	110	Select internal clock divided by 64.	
	111	Select corresponding external clock (cka_in) for serial clock.	

ASCI Status Register		(STAT0)	(Address = 0x0004)
		(STAT1)	(Address = 0x0005)
Bit(s)	Value	Description (Async mode only)	
7	0	The receive buffer is empty.	
	1	The receive buffer is full. This bit is cleared by reading RDR, while the corresponding dcdb input is High and during IOSTOP mode. If enabled, an interrupt is requested while this bit is set. The interrupt is cleared when the receive buffer is empty.	
6	0	The receive buffer was not overrun.	
	1	The receive buffer was overrun. This bit is cleared while the corresponding dcdb input is High, during IOSTOP mode, and by writing a 0 to bit 3 of CNTLA.	
5	0	The received byte did not have a parity error.	
	1	The received byte had a parity error. This bit is cleared while the corresponding dcdb input is High, during IOSTOP mode, and by writing a 0 to bit 3 of CNTLA.	
4	0	The received byte did not have a framing error.	
	1	The received byte had a framing error. This bit is cleared while the corresponding dcdb input is High, during IOSTOP mode, and by writing a 0 to bit 3 of CNTLA.	
3	0	Disable receive interrupts.	
	1	Enable receive interrupts.	
2 (rd-only)	0	The corresponding dcdb input is Low.	
	1	The corresponding dcdb input is High.	
1	0	The transmit buffer is full.	
	1	The transmit buffer is empty. This bit is set by writing TDR, while the corresponding ctsb input is High and during IOSTOP mode. If enabled, an interrupt is requested while this bit is set. The interrupt is cleared when the transmit buffer is full.	
0	0	Disable transmit interrupts.	
	1	Enable transmit interrupts.	

ASCI Transmit Data Register		(TDR0)	(Address = 0x0006)
		(TDR1)	(Address = 0x0007)
Bit(s)	Value	Description	
7:0	Read	Returns the contents of the transmit buffer, independent of whether the byte has been transmitted or not.	
	Write	Loads the transmit buffer with a data byte for transmission.	

ASCI Receive Data Register		(RDR0)	(Address = 0x0008)
		(RDR1)	(Address = 0x0009)
Bit(s)	Value	Description	
7:0	Read	Returns the contents of the receive buffer.	
	Write	Loads the receive buffer with a byte, but only if the buffer is empty.	

ASCI Extension Control Register		(ECNTL0)	(Address = 0x0012)
		(ECNTL1)	(Address = 0x0013)
Bit(s)	Value	Description	
7	0	Normal operation for receive data interrupt: interrupt while receive buffer full.	
	1	No interrupt while receive buffer full. Used with receive DMA transfers.	
6	0	Enable corresponding dcdb as a receive auto-enable.	
	1	Disable corresponding dcdb as a receive auto-enable.	
5	0	Enable corresponding ctsb as a transmit auto-enable.	
	1	Disable corresponding ctsb as a transmit auto-enable.	
4		This bit is reserved and should always be written as zero.	
3	0	Disable 16-bit baud rate generator counter.	
	1	Enable 16-bit baud rate generator counter.	
2	0	Disable Break feature.	
	1	Enable Break feature (detection and generation of Break).	
1 (rd-only)	0	No Break detected.	
	1	Break Detected. This bit remains set for the duration of the Break condition.	
0	0	No effect.	
	1	Send Break, by forcing the corresponding txa signal Low. Software must time the duration of the break.	

ASCI Time Constant Low Register		(TCL0)	(Address = 0x001A)
		(TCL1)	(Address = 0x001C)
Bit(s)	Value	Description	
7:0		Eight LSBs of the divider that generates the serial clock for the channel. This divider is not used unless enabled in the ASCII Extension Control Register.	

ASCI Time Constant High Register		(TCH0)	(Address = 0x001B)
		(TCH1)	(Address = 0x001D)
Bit(s)	Value	Description	
7:0		Eight MSBs of the divider that generates the serial clock for the channel. This divider is not used unless enabled in the ASCI Extension Control Register. The divider counts with a period of $2^*({TCH,TCL}+2)$.	

Prog. Reload Timer

The Programmable Reload Timer (PRT) provides two identical, independent 16-bit timers for pulse or waveform generation.

Features:

- 16-bit down-counter with automatic reload on terminal count
- clocked at **clk**/20 rate
- **tout** (Timer Out) signal can be set/cleared/toggled by timer terminal count

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
Timer Data Register 0L	TMDR0L	0x000C	R/W	11111111
Timer Data Register 0H	TMDR0H	0x000D	R/W	11111111
Timer Reload Register 0L	RLDR0L	0x000E	R/W	11111111
Timer Reload Register 0H	RLDR0H	0x000F	R/W	11111111
Timer Control Register	TCR	0x0010	R/W	00000000
Timer Data Register 1L	TMDR1L	0x0014	R/W	11111111
Timer Data Register 1H	TMDR1H	0x0015	R/W	11111111
Timer Reload Register 1L	RLDR1L	0x0016	R/W	11111111
Timer Reload Register 1H	RLDR1H	0x0017	R/W	11111111

Register Descriptions

Timer Control Register		(TCR)	(Address = 0x0010)
Bit(s)	Value	Description	
7 (rd-only)	0	This bit is cleared by first reading the TCR, followed by a reading either TMDR1L or TMDR 1H.	
	1	Timer 1 has decremented to 0x0000. If enabled, an interrupt will be requested while this bit is set.	
6 (rd-only)	0	This bit is cleared by first reading the TCR, followed by reading either TMDR0L or TMDR 0H.	
	1	Timer 0 has decremented to 0x0000. If enabled, an interrupt will be requested while this bit is set.	
5	0	Disable Timer 1 interrupt.	
	1	Enable Timer 1 interrupt.	
4	0	Disable Timer 0 interrupt.	
	1	Enable Timer 0 interrupt.	
3:2	00	Terminal count does not affect tout signals.	
	01	tout signal toggles on corresponding count reaches 0x0000.	
	10	tout signal set Low on corresponding count reaches 0x0000.	
	11	tout signal set High on corresponding count reaches 0x0000.	
1	0	Disable Timer 1 counting.	
	1	Enable Timer 1 counting.	
0	0	Disable Timer 0 counting.	
	1	Enable Timer 0 counting.	

Timer Data Register Low		(TMDR0L)	(Address = 0x000C)
		(TMDR1L)	(Address = 0x0014)
Bit(s)	Value	Description	
7:0	Read	Returns the LSB of the current count value. Reading this register latches the MSB for reading in the TMDRH, so the count should always be read LSB-first.	
	Write	Loads the LSB of the counter. Used to set the initial value for the counter. The counter will automatically reload from the RLDR after reaching a count of 0x0000.	

Timer Data Register High		(TMDR0H)	(Address = 0x000D)
		(TMDR1H)	(Address = 0x0015)
Bit(s)	Value	Description	
7:0	Read	Returns the MSB of the current count value. Reading the TMDRL automatically latches the MSB for reading in this register to guarantee a valid 16-bit value.	
	Write	Loads the MSB of the counter. Used to set the initial value for the counter. The counter will automatically reload from the RLDR after reaching a count of 0x0000.	

Timer Reload Register Low		(RLDR0L)	(Address = 0x000E)
		(RLDR1L)	(Address = 0x0016)
Bit(s)	Value	Description	
7:0		LSB of the reload value for the counter. The reload value is automatically loaded into the counter after the count of 0x0000.	

Timer Reload Register High		(RLDR0H)	(Address = 0x000F)
		(RLDR1H)	(Address = 0x0017)
Bit(s)	Value	Description	
7:0		MSB of the reload value for the counter. The reload value is aut	

Clocked Serial I/O

The Clocked Serial I/O (CSIO) port provides simple high-speed half-duplex serial communications.

Features:

- Half-duplex, 8 bits/character
- Common transmit/receive data register
- External clock input or dedicated divider
- Separate **cks_in** (CSIO Clock In) and **cks_out** (CSIO Clock Out) signals.

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
CSIO Control/Status Register	CNTR	0x000A	R/W	0000x111
CSIO Transmit/Receive Register	TRDR	0x000B	R/W	00000000

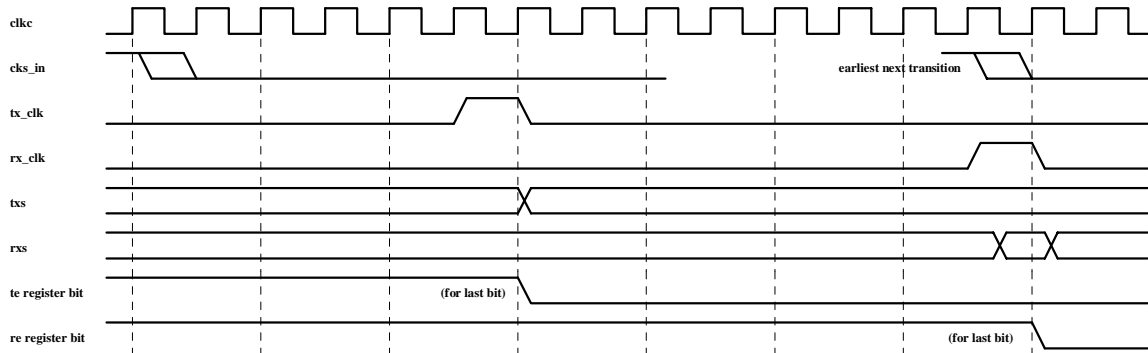
Register Descriptions

CSIO Control/Status Register (CNTR) (Address = 0x000A)		
Bit(s)	Value	Description
7 (rd-only)	0	Transmit or receive operation started. This bit is cleared by a read or write of the TRDR. This bit is also cleared in IOSTOP mode.
	1	Transmit or receive operation complete. If enabled, an interrupt is requested while this bit is set.
6	0	Disable CSIO interrupt.
	1	Enable CSIO interrupt.
5:4	00	Transmit or receive operation complete. These bits are cleared by the CSIO to indicate that the transmit or receive operation is complete. These bits are also cleared in IOSTOP mode.
	01	Start receive operation.
	10	Start transmit operation.
	11	This bit combination is invalid and will be ignored if written.
3		This bit is reserved and should always be written as zero.
2:0	000	Internal clock, clk /20 rate.
	001	Internal clock, clk /40 rate.
	010	Internal clock, clk /80 rate.
	011	Internal clock, clk /160 rate.
	100	Internal clock, clk /320 rate.
	101	Internal clock, clk /640 rate.
	110	Internal clock, clk /1280 rate.
	111	External clock (must be slower than clk /20).

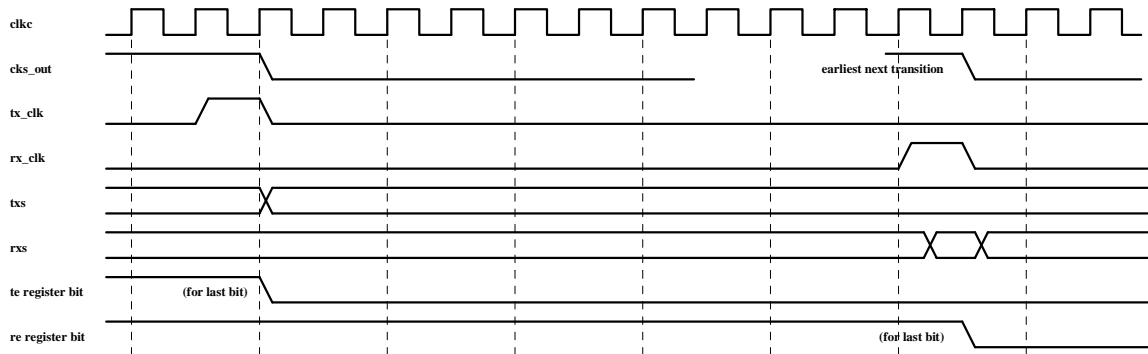
CSIO Transmit/Receive Data Register (TRDR) (Address = 0x000B)		
Bit(s)	Value	Description
7:0	Read	Returns the received byte. This register is not buffered, so data is not valid until the receive operation is signalled complete. Note that the rxs input is still shifted in during a transmit operation.
	Write	Loads a byte for transmission. This register is not buffered, so data written to this register while a receive or transmit operation is in progress will corrupt the receive or transmit data.

Timing

The CSIO port uses a clock that is synchronized to the the **clk** signal, which means the the clock signal is not used directly for either transmit or receive data. The falling edge of the CSIO clock (whether external or internal) is used to both send the transmit data and sample the receive data. The figure below shows the timing for the external clock case. Note that the CSIO clock is limited to a maximum rate of **clk/13** in this case.



The timing for the internal clock case is shown in the figure below. Note that the CSIO clock is limited to a maximum rate of **clk/11** in this case. Note that the register selection available to the programmer limit the maximum rate to **clk/20**.



System Functions

The System Functions block contains three of the four system-level functions present in the original Z80180. The Memory Refresh function is not included, as modern DRAMs usually contain internal refresh circuitry.

Features

- Relocatable on-chip I/O (starting at address 0x0000, 0x0040, 0x0080 or 0x00C0)
- Free Running Counter counts down at **clk**/10 rate
- I/O Stop mode

Registers

Register Name	Mnemonic	I/O address	R/W	Reset
Free Running Counter	FRC	0x0018	R/W	11111111
I/O Control Register	ICR	0x003F	R/W	111xxxxx

Register Descriptions

Free Running Counter (FRC) (Address = 0x0018)		
Bit(s)	Value	Description
7:0		Current value of Free Running Counter. This counter decrements at a clk /10 rate.

I/O Control Register (BBR) (Address = 0x0039)		
Bit(s)	Value	Description
7:6	00	Internal I/O address range is 0x0000 - 0x003F.
	01	Internal I/O address range is 0x0040 - 0x007F.
	10	Internal I/O address range is 0x0080 - 0x00BF.
	11	Internal I/O address range is 0x00C0 - 0x00FF.
5	0	Normal I/O operation.
	1	I/O Stop mode. This mode disables the receiver, transmitter and BRG in the SCI channels, immediately halts the CSIO port, and immediately disable the timers in the PRT and FRC.
4:0		These bits are reserved and will always be read as ones.

Register Addresses

The table below lists all of the on-chip I/O registers in the Y90-180 along with their mnemonic, I/O address and reset state.

Register Name	Mnemonic	I/O address	R/W	Reset
ASCI Control Register A, Channel 0	CNTLA0	0x0000	R/W	0001x000
ASCI Control Register A, Channel 1	CNTLA1	0x0001	R/W	0001x000
ASCI Control Register B, Channel 0	CNTLB0	0x0002	R/W	x0x00111
ASCI Control Register B, Channel 1	CNTLB1	0x0003	R/W	x0x00111
ASCI Status Register, Channel 0	STAT0	0x0004	R/W	00000xx0
ASCI Status Register, Channel 1	STAT1	0x0005	R/W	00000xx0
ASCI Transmit Data Register, Channel 0	TDR0	0x0006	R/W	xxxxxxxx
ASCI Transmit Data Register, Channel 1	TDR1	0x0007	R/W	xxxxxxxx
ASCI Receive Data Register, Channel 0	RDR0	0x0008	R/W	xxxxxxxx
ASCI Receive Data Register, Channel 1	RDR1	0x0009	R/W	xxxxxxxx
CSIO Control/Status Register	CNTR	0x000A	R/W	0000x111
CSIO Transmit/Receive Register	TRDR	0x000B	R/W	00000000
Timer Data Register 0L	TMDR0L	0x000C	R/W	11111111
Timer Data Register 0H	TMDR0H	0x000D	R/W	11111111
Timer Reload Register 0L	RLDR0L	0x000E	R/W	11111111
Timer Reload Register 0H	RLDR0H	0x000F	R/W	11111111
Timer Control Register	TCR	0x0010	R/W	00000000
Reserved		0x0011		
ASCI Extension Control Register, Channel 0	ECNTL0	0x0012	R/W	00000000
ASCI Extension Control Register, Channel 1	ECNTL1	0x0013	R/W	00000000
Timer Data Register 1L	TMDR1L	0x0014	R/W	11111111
Timer Data Register 1H	TMDR1H	0x0015	R/W	11111111
Timer Reload Register 1L	RLDR1L	0x0016	R/W	11111111
Timer Reload Register 1H	RLDR1H	0x0017	R/W	11111111
Free Running Counter	FRC	0x0018	R/W	11111111
Reserved		0x0019		

ASCI Time Constant Low Register, Channel 0	TCL0	0x001A	R/W	00000000
ASCI Time Constant High Register, Channel 0	TCH0	0x001B	R/W	00000000
ASCI Time Constant Low Register, Channel 1	TCL1	0x001C	R/W	00000000
ASCI Time Constant High Register, Channel 1	TCH1	0x001D	R/W	00000000
Reserved		0x001E		
Reserved		0x001F		
DMA Source Address Register 0 Low	SAR0L	0x0020	R/W	00000000
DMA Source Address Register 0 High	SAR0H	0x0021	R/W	00000000
DMA Source Address Register 0 Page	SAR0P	0x0022	R/W	00000000
DMA Destination Address Register 0 Low	DAR0L	0x0023	R/W	00000000
DMA Destination Address Register 0 High	DAR0H	0x0024	R/W	00000000
DMA Destination Address Register 0 Page	DAR0P	0x0025	R/W	00000000
DMA Byte Count Register 0 Low	BCR0L	0x0026	R/W	00000000
DMA Byte Count Register 0 High	BCR0H	0x0027	R/W	00000000
DMA Memory Address Register 1 Low	MAR1L	0x0028	R/W	00000000
DMA Memory Address Register 1 High	MAR1H	0x0029	R/W	00000000
DMA Memory Address Register 1 Page	MAR1P	0x002A	R/W	00000000
DMA I/O Address Register 1 Low	IAR1L	0x002B	R/W	00000000
DMA I/O Address Register 1 High	IAR1H	0x002C	R/W	00000000
DMA I/O Address Register 1 Page	IAR1P	0x002D	R/W	00000000
DMA Byte Count Register 1 Low	BCR1L	0x002E	R/W	00000000
DMA Byte Count Register 1 High	BCR1H	0x002F	R/W	00000000
DMA Status Register	DSTAT	0x0030	R/W	00000000
DMA Mode Register	DMODE	0x0031	R/W	00000000
DMA/Wait Control Register	DCNTL	0x0032	R/W	11110000
Interrupt Vector Low Register	IL	0x0033	R/W	00000000
Interrupt/Trap Control Register	ITC	0x0034	R/W	00111001
Reserved		0x0035		
Reserved (Refresh Control not implemented)		0x0036		
Reserved		0x0037		
MMU Common Base Register	CBR	0x0038	R/W	00000000
MMU Bank Base Register	BBR	0x0039	R/W	00000000
MMU Common/Bank Area Register	CBAR	0x003A	R/W	11110000
Reserved		0x003B		
Reserved		0x003C		
Reserved		0x003D		
Reserved		0x003E		
I/O Control Register	ICR	0x003F	R/W	00000000

Top Level Verilog Code

The Verilog code for the y90-180 is shown below to illustrate how the individual modules connect.

```

/*****
**
** COPYRIGHT (C) 2010, SYSTEMYDE INTERNATIONAL CORPORATION, ALL RIGHTS RESERVED
**
** z180-style mpu
** Rev 0.0 04/12/2010
**
*****/
module y90_180 (cka0_out, cka1_out, cks_ext, cks_out, dma_ack, halt_tran, iack_tran,
io_addr_out, io_data_out, io_read, io_strobe, io_tran, ivec_rd, mem_addr_out,
mem_data_out, mem_rd, mem_tran, mem_wr, reti_tran, rts0b, rts1b, sleep_tran,
t1, tend0b, tend1b, tout0, tout1, txa0, txal, txs, cka0_in, cka1_in, cks_in,
clearb, clkc, cts0b, cts1b, dcd0b, dcd1b, dma_req, dreq0b, dreq1b, en_prftch,
int0_req, int1_req, int2_req, io_data_in, ivec_data_in, mem_data_in, nmi_req,
resetb, rxa0, rxal, rxs, wait_req);

input      cka0_in;      /* asci 0 clock input */
input      cka1_in;      /* asci 0 clock input */
input      cks_in;      /* csio clock input */
input      clearb;      /* master (test) reset */
input      clkc;        /* main cpu clock */
input      cts0b;       /* cts 0 input (active low) */
input      cts1b;       /* cts 1 input (active low) */
input      dcd0b;       /* dcd 0 input (active low) */
input      dcd1b;       /* dcd 1 input (active low) */
input      dma_req;     /* dma request */
input      dreq0b;      /* dma0 external request */
input      dreq1b;      /* dma1 external request */
input      en_prftch;   /* prefetch enable */
input      int0_req;    /* ext interrupt 0 */
input      int1_req;    /* ext interrupt 1 */
input      int2_req;    /* ext interrupt 2 */
input      nmi_req;     /* nmi request */
input      resetb;     /* internal (user) reset */
input      rxa0;        /* asci 0 data input */
input      rxal;        /* asci 1 data input */
input      rxs;         /* csio data input */
input      wait_req;    /* wait request */
input      [7:0] io_data_in; /* i/o input data bus */
input      [7:0] ivec_data_in; /* int vector bus */
input      [7:0] mem_data_in; /* memory input bus */
output     cka0_out;    /* asci 0 clock output */
output     cka1_out;    /* asci 1 clock output */
output     cks_ext;     /* csio clock external */
output     cks_out;     /* csio clock output */
output     dma_ack;     /* dma acknowledge */
output     halt_tran;   /* halt transaction */
output     iack_tran;   /* interrupt acknowledge transaction */
output     io_read;    /* i/o read enable */
output     io_strobe;   /* i/o data strobe */
output     io_tran;     /* i/o transaction */
output     ivec_rd;     /* interrupt vector enable */
output     mem_rd;      /* memory read enable */
output     mem_tran;    /* memory transaction */
output     mem_wr;      /* memory write enable */
output     reti_tran;   /* return from interrupt transaction */
output     rts0b;       /* rts 0 output */

```

```

output      rts1b;          /* rts 1 output          */
output      sleep_tran;    /* sleep transaction     */
output      t1;           /* first clock of transaction */
output      tend0b;       /* dma0 end pulse        */
output      tend1b;       /* dma1 end pulse        */
output      tout0;        /* prt 0 timer output    */
output      tout1;        /* prt 1 timer output    */
output      txa0;         /* asci 0 data output    */
output      txal;         /* asci 1 data output    */
output      txs;          /* csio data output      */
output [7:0] io_data_out;  /* i/o output data bus   */
output [7:0] mem_data_out; /* memory output data bus */
output [15:0] io_addr_out; /* i/o address bus      */
output [19:0] mem_addr_out; /* memory address bus    */

/*****
/*
/* signal declarations
/*
/*****
wire      asci0_int;      /* asci 0 interrupt      */
wire      asci1_int;      /* asci 1 interrupt      */
wire      asci0_rxreq;    /* asci 0 rx request     */
wire      asci0_txreq;    /* asci 0 tx request     */
wire      asci1_rxreq;    /* asci 1 rx request     */
wire      asci1_txreq;    /* asci 1 tx request     */
wire      cka0_out;       /* asci 0 clock output   */
wire      cka1_out;       /* asci 1 clock output   */
wire      cks_ext;        /* csio clock external   */
wire      cks_out;        /* csio clock output     */
wire      clk10c;         /* clk/10 clock enable   */
wire      clk10cs;        /* clk/10 symmetric     */
wire      clk20c;         /* clk/20 clock enable   */
wire      clk30c;         /* clk/30 clock enable   */
wire      clk30cs;        /* clk/30 symmetric     */
wire      csio_int;       /* csio interrupt        */
wire      dma_ack;        /* dma acknowledge (sys) */
wire      dma_ack_s;      /* dma acknowledge       */
wire      dma_req;        /* dma request           */
wire      dma_req_s;      /* dma request (sys)     */
wire      dma0_int;       /* dma 0 interrupt       */
wire      dma1_int;       /* dma 1 interrupt       */
wire      ftch_tran;      /* inst fetch transaction */
wire      frc_imd2;       /* force int mode 2     */
wire      halt_tran;      /* halt transaction      */
wire      iack_tran;      /* int ack transaction   */
wire      inst2_trap;     /* illegal 2-byte instruction */
wire      inst3_trap;     /* illegal 3-byte instruction */
wire      int_req;        /* interrupt request     */
wire      io_read;        /* i/o read enable       */
wire      io_read_c;      /* i/o read enable (cpu) */
wire      io_read_d;      /* i/o read enable (dma) */
wire      io_stop;        /* i/o stop mode         */
wire      io_strobe;      /* i/o data strobe       */
wire      io_strobe_c;    /* i/o data strobe (cpu) */
wire      io_strobe_d;    /* i/o data strobe (dma) */
wire      io_tran;        /* i/o transaction       */
wire      io_tran_c;      /* i/o transaction (cpu) */
wire      io_tran_d;      /* i/o transaction (dma) */
wire      ivec_rd;        /* interrupt vector enable */
wire      ld_if1;         /* load inst byte 1     */
wire      ld_mem_addr;    /* update memory address */
wire      mem_rd;         /* memory read enable    */
wire      mem_rd_c;       /* memory read enable (cpu) */
wire      mem_rd_d;       /* memory read enable (dma) */
wire      mem_tran;       /* memory transaction    */
wire      mem_tran_c;     /* memory transaction (cpu) */
wire      mem_tran_d;     /* memory transaction (dma) */
wire      mem_wr;         /* memory write enable   */
wire      mem_wr_c;       /* memory write enable (cpu) */
wire      mem_wr_d;       /* memory write enable (dma) */
wire      output_inh;     /* disable cpu outputs   */
wire      prt0_int;       /* prt ch0 interrupt     */
wire      prt1_int;       /* prt ch1 interrupt     */
wire      rd_peri;        /* peripheral read       */
wire      reti_tran;      /* reti transaction     */
wire      rts0b;         /* rts 0 output          */
wire      rts1b;         /* rts 1 output          */

```

```

wire          sleep_tran;          /* sleep transaction          */
wire          t1;                   /* first clock of transaction */
wire          t1_c;                 /* first clock of trans (cpu)  */
wire          t1_d;                 /* first clock of trans (dma)  */
wire          tend0b;               /* dma 0 end pulse            */
wire          tend1b;               /* dma 1 end pulse            */
wire          tout0;                /* prt 0 timer output         */
wire          tout1;                /* prt 1 timer output         */
wire          txa0;                  /* asci 0 data output         */
wire          txa1;                  /* asci 1 data output         */
wire          txs;                   /* csio data output           */
wire          wait_req_s;           /* wait request (sys)         */
wire          wait_st;              /* wait state identifier       */
wire          wr_peri;              /* peripheral write            */
wire [2:0]    inten_reg;            /* interrupt enable bits       */
wire [3:0]    log_addr_page;        /* logical address page        */
wire [7:0]    asci_data;            /* asci read data              */
wire [7:0]    csio_data;            /* csio read data              */
wire [7:0]    dma_data;             /* dma read data               */
wire [7:0]    int_data;             /* int read data               */
wire [7:0]    io_data_in_s;         /* i/o input data bus (sys)    */
wire [7:0]    io_data_out;          /* i/o output data bus         */
wire [7:0]    io_data_out_c;        /* i/o output data bus (cpu)   */
wire [7:0]    io_data_out_d;        /* i/o output data bus (dma)   */
wire [7:0]    ivec_data_in_s;       /* int vector bus (sys)        */
wire [7:0]    mem_data_out;         /* memory output data bus      */
wire [7:0]    mem_data_out_c;       /* memory output data bus (cpu)*/
wire [7:0]    mem_data_out_d;       /* memory output data bus (dma)*/
wire [7:0]    mmu_data;             /* mmu read data               */
wire [7:0]    prt_data;             /* prt read data               */
wire [15:0]   addr_reg_in;          /* processor logical address   */
wire [15:0]   io_addr_out;          /* i/o address bus             */
wire [15:0]   io_addr_out_c;        /* i/o address bus (cpu)       */
wire [15:0]   io_addr_out_d;        /* i/o address bus (dma)       */
wire [19:0]   mem_addr_out;         /* memory address bus          */
wire [19:0]   mem_addr_out_c;       /* memory address bus (cpu)     */
wire [19:0]   mem_addr_out_d;       /* memory address bus (dma)     */

/*****
/*
/* processor
/*
/*****
y90_core CPU      (.addr_reg_in(addr_reg_in), .burst_done(), .ctr_reg(),
                  .dma_ack(dma_ack_s), .fault_detect(), .ftch_tran(ftch_tran),
                  .halt_tran(halt_tran), .iack_tran(iack_tran), .imd2_reg(),
                  .inst2_trap(inst2_trap), .inst3_trap(inst3_trap),
                  .io_addr_out(io_addr_out_c), .io_data_out(io_data_out_c),
                  .io_read(io_read_c), .io_strobe(io_strobe_c), .io_tran(io_tran_c),
                  .ivec_rd(ivec_rd), .ld_if1(ld_if1), .ld_init(),
                  .ld_mem_addr(ld_mem_addr),
                  .mem_addr_out({log_addr_page, mem_addr_out_c[11:0]}),
                  .mem_data_out(mem_data_out_c), .mem_rd(mem_rd_c),
                  .mem_tran(mem_tran_c),
                  .mem_wr(mem_wr_c), .mmu_msb(), .mmu_swap(),
                  .mwr_tran(), .output_inh(output_inh), .reti_tran(reti_tran),
                  .sleep_tran(sleep_tran), .stk_tran(), .t1(t1_c), .wait_st(wait_st),
                  .wdt_arm(), .wdt_hit(), .wr_brst(), .clearb(clearb), .clkc(clkc),
                  .dma_req(dma_req_s), .en_prftch(en_prftch), .frc_imd2(frc_imd2),
                  .int_req(int_req), .io_data_in(io_data_in_s),
                  .ivec_data_in(ivec_data_in_s), .mem_data_in(mem_data_in),
                  .nmi_req(nmi_req), .resetb(resetb), .wait_req(wait_req_s) );

/*****
/*
/* memory management unit
/*
/*****
mmu_seg MMU      (.mmu_addr_out(mem_addr_out_c[19:12]), .mmu_data(mmu_data),
                  .addr_reg_in(addr_reg_in[15:12]), .clkc(clkc),
                  .io_addr_out(io_addr_out[5:0]), .io_data_out(io_data_out),
                  .ld_mem_addr(ld_mem_addr), .output_inh(output_inh),
                  .resetb(resetb), .wr_peri(wr_peri) );

/*****
/*
/* system management
/*

```

```

/*****/
sys_180 SYS      ( .clk10c(clk10c), .clk10cs(clk10cs), .clk20c(clk20c), .clk30c(clk30c),
                 .clk30cs(clk30cs), .inten_reg(inten_reg), .io_stop(io_stop),
                 .rd_peri(rd_peri), .io_data_in_s(io_data_in_s),
                 .wait_req_s(wait_req_s), .wr_peri(wr_peri), .ascii_data(ascii_data),
                 .clkc(clkc), .csio_data(csio_data), .dma_data(dma_data),
                 .iack_tran(iack_tran), .inst2_trap(inst2_trap),
                 .inst3_trap(inst3_trap), .int_data(int_data),
                 .io_addr_out(io_addr_out), .io_data_in(io_data_in),
                 .io_data_out(io_data_out), .io_read(io_read), .io_strobe(io_strobe),
                 .io_tran(io_tran), .mem_tran(mem_tran), .mmu_data(mmu_data),
                 .prt_data(prt_data), .resetb(resetb), .tl(tl), .wait_req(wait_req) );

/*****/
/*
/* dma
/*
/*****/
dma_180 DMA      ( .bus_ack(dma_ack), .dma_data(dma_data), .dma_req(dma_req_s),
                 .dma0_int(dma0_int), .dmal_int(dmal_int),
                 .io_addr_out_d(io_addr_out_d), .io_data_out_d(io_data_out_d),
                 .io_read(io_read_d), .io_strobe(io_strobe_d), .io_tran(io_tran_d),
                 .mem_addr_out_d(mem_addr_out_d), .mem_data_out_d(mem_data_out_d),
                 .mem_rd(mem_rd_d), .mem_tran(mem_tran_d), .mem_wr(mem_wr_d), .tl(tl_d),
                 .tend0b(tend0b), .tendlb(tendlb), .ascii0_rxreq(ascii0_rxreq),
                 .ascii0_txreq(ascii0_txreq), .ascil_rxreq(ascil_rxreq),
                 .ascil_txreq(ascil_txreq), .bus_req(dma_req), .clearb(clearb),
                 .clkc(clkc), .dma_ack(dma_ack_s), .dreq0b(dreq0b), .dreq1b(dreq1b),
                 .io_addr_out(io_addr_out[5:0]), .io_data_in(io_data_in),
                 .io_data_out(io_data_out), .mem_data_in(mem_data_in),
                 .nmi_req(nmi_req), .resetb(resetb), .sleep_tran(sleep_tran),
                 .wait_req(wait_req_s), .wr_peri(wr_peri) );

assign io_addr_out = io_addr_out_c | io_addr_out_d;
assign io_data_out = io_data_out_c | io_data_out_d;
assign io_read     = io_read_c     | io_read_d;
assign io_strobe   = io_strobe_c   | io_strobe_d;
assign io_tran     = io_tran_c     | io_tran_d;
assign mem_addr_out = mem_addr_out_c | mem_addr_out_d;
assign mem_data_out = mem_data_out_c | mem_data_out_d;
assign mem_rd      = mem_rd_c      | mem_rd_d;
assign mem_tran    = mem_tran_c    | mem_tran_d;
assign mem_wr      = mem_wr_c      | mem_wr_d;
assign tl          = tl_c          | tl_d;

/*****/
/*
/* interrupt control
/*
/*****/
int_180 INT      ( .frc_imd2(frc_imd2), .int_data(int_data), .int_req(int_req),
                 .ivec_data_out(ivec_data_in_s), .ascii0_int(ascii0_int),
                 .ascil_int(ascil_int), .clkc(clkc), .csio_int(csio_int),
                 .dma0_int(dma0_int), .dmal_int(dmal_int), .iack_tran(iack_tran),
                 .int0_req(int0_req), .intl_req(intl_req), .int2_req(int2_req),
                 .inten_reg(inten_reg), .io_addr_out(io_addr_out[5:0]),
                 .io_data_out(io_data_out), .ivec_data_in(ivec_data_in),
                 .ivec_rd(ivec_rd), .prt0_int(prt0_int), .prt1_int(prt1_int),
                 .resetb(resetb), .reti_tran(reti_tran), .tl(tl), .wait_st(wait_st),
                 .wr_peri(wr_peri) );

/*****/
/*
/* async serial comm interface
/*
/*****/
ascii_180 ASCII  ( .ascii_data(ascii_data), .ascii0_int(ascii0_int),
                 .ascii0_rxreq(ascii0_rxreq), .ascii0_txreq(ascii0_txreq),
                 .ascil_int(ascil_int), .ascil_rxreq(ascil_rxreq),
                 .ascil_txreq(ascil_txreq), .cka0_out(cka0_out), .ckal_out(ckal_out),
                 .rts0b(rts0b), .rts1b(rts1b), .txa0(txa0), .txal(txal),
                 .cka0_in(cka0_in), .ckal_in(ckal_in), .clk10c(clk10c),
                 .clk10cs(clk10cs), .clk30c(clk30c), .clk30cs(clk30cs), .clkc(clkc),
                 .cts0b(cts0b), .cts1b(cts1b), .dcd0b(dcd0b), .dcd1b(dcd1b),
                 .io_addr_out(io_addr_out[5:0]), .io_data_out(io_data_out),
                 .io_stop(io_stop), .rd_peri(rd_peri), .resetb(resetb), .rxa0(rxa0),
                 .rxal(rxal), .wr_peri(wr_peri) );

```

```

/*****
/*
/* programmable reload timer
/*
/*****
prt_180 PRT      ( .prt_data(prt_data), .prt0_int(prt0_int), .prt1_int(prt1_int),
                  .tout0(tout0), .tout1(tout1), .clk20c(clk20c), .clkc(clkc),
                  .io_addr_out(io_addr_out[5:0]), .io_data_out(io_data_out),
                  .rd_peri(rd_peri), .resetb(resetb), .wr_peri(wr_peri) );

/*****
/*
/* clocked serial i/o
/*
/*****
csio_180 CSIO    ( .cks_ext(cks_ext), .cks_out(cks_out), .csio_data(csio_data),
                  .csio_int(csio_int), .txs(txs), .cks_in(cks_in), .clk10c(clk10c),
                  .clkc(clkc), .io_addr_out(io_addr_out[5:0]), .io_data_out(io_data_out),
                  .io_stop(io_stop), .rd_peri(rd_peri), .resetb(resetb), .rxs(rxs),
                  .wr_peri(wr_peri) );

endmodule

```

