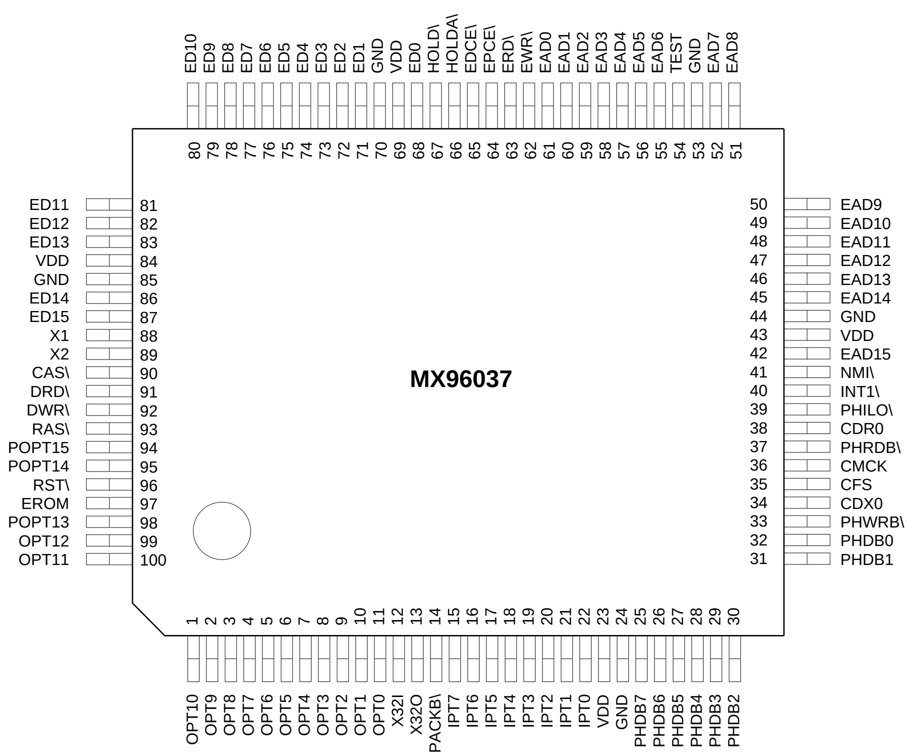
FEATURES

- 16-bit, 47/54/65 ns cycle (21/18/15 MIPs) DSP controller.
- 16x16 multiplier, one cycle multiply-accumulate.
- 32-bit ALU and 16-bit auxiliary ALU (ARAU) work in parallel.
- 8 auxiliary registers for indirect addressing work with ARAU.
- 16-level hardware stack and nestable interrupt support.
- 32-bit barrel shifter.
- 8-instruction looped up to 128 times capability.
- Block program move.
- 64k words program ROM space, 18k words may be internal.
- External ROM option may replace internal 18K for fast prototyping.
- 64k words SRAM space, 2048 words internal.
- 32 internal IO address.
- 1 independent interrupt pin, 1 NMI pin.
- 8 input pins.
- 8 bi-direction I/O pins.
- 16 output pins.
- Hold or slow system clock for power management.
- 1 ms system tick timer for system timing.
- One Codec interface.
- Built-in DRAM Controller;1G addressing space, with 1/4/8/16 data bit interface support.
- Single 5V supply, 100 pins PQFP

PIN CONFIGURATION

100 PQFP



PIN DESCRIPTIONS

A. DSP BASIC (22 PINS)

SYMBOL	PIN TYPE	PIN NUMBER	DESCRIPTION
VDD		23, 43, 69, 84	5V power source
GND		24, 44, 53, 70, 85	Ground
X1		88	Crystal input
X2		89	Crystal output
RST\	I	96	Power-on Reset, Schmite-triggered
PACKB\(XF\)	OA	14	If UPMODX=1 External flag, this pin can be directly written by one DSP instruction. Default inactive (5V output). UPMODX=0, ACKNOWLEDGE to host, data is ready in CMR.
HOLD\	I	67	Hold DSP clock down and release bus
HOLDA\	OA	66	Ack to HOLD\ signal
EROM	I	97	Disable internal ROM; use external ROM only.
NMI\	I	41	Non maskable interrupt pin.
INT1\	I	40	Interrupt pin
X32O	OA	13	32k Crystal output.
X32I		12	32k Crystal input.
TEST	I	54	Connect to VDD (for test only).

PIN DESCRIPTIONS *(Continued)*

B. DSP EXTERNAL MEMORY (41 PINS)

SYMBOL	PIN TYPE	PIN NUMBER	DESCRIPTION
EAD0-EAD15	OB	61-55, 52-45, 42	DSP IO/RAM/ROM external address bus. EAD0-EAD14 are for DRAM address also.
ED0-ED15	I/OB	68, 71-83, 86-87	DSP IO/RAM/ROM/DRAM external data bus.
EDCE\	OB	65	External data chip enable.
EPCE\	OB	64	External program chip enable.
ERD\	OB	63	SRAM/ROM/IO external read.
EWR\	OB	62	SRAM/ROM/IO external write.
CAS\	OB	90	DRAM column address select.
RAS\	OB	93	DRAM row address select.
DRD\	OB	91	DRAM read.
DWR\	OB	92	DRAM write.

MP INTERFACE

SYMBOL	PIN TYPE	PIN NUMBER	DESCRIPTION
PHILO\	I	39	High- or low-byte select (UPMODX =0). Output port OPT 18 (UPMODX =1).
PHRDB\	I	37	Host read (UPMODX =0.) Output port OPT 17 (UPMODX=1).
PHWR\	I	33	Host write (UPMODX =0). Output port OPT 16 (UPMODX =1).
PHDB(7:0)	B	25~32	Host data bus (UPMODX =0). Bidirectional IO port (UPMODX =1).

PIN DESCRIPTIONS *(Continued)*

D. CODEC (6 PINS)

SYMBOL	PIN TYPE	PIN NUMBER	DESCRIPTION
CFS	OB	35	Codec frame sync, 8 KHz. (6KHz)
CMCK	OB	36	Codec master clock, 1.536 MHz (default output).
CDX0	OA	34	Codec data transmit
CDR0	I	38	Codec data receive

E. OPT : Output port (13 PINS)

SYMBOL	PIN TYPE	PIN NUMBER	DESCRIPTION
OPT0-OPT15	OC	1-11,94,95,98 99, 100	Output to pin, all output values are registered and may be read back when read by 'IN' instruction.

F. IPT : Input port (8 PINS)

SYMBOL	PIN TYPE	PIN NUMBER	DESCRIPTION
IPT7-IPT0	I	15-22	Input port. IPT0-IPT3 with internal pull high

G. BIO : Bi-direction I/O (8 PINS)

SYMBOL	PIN TYPE	PIN NUMBER	DESCRIPTION
BIO0-BIO7	I/OB	25-32	Input/output port. Direction is controlled by BIO15-BIO8, (see BIOR).

NOTE: OA = 2mA, OB = 4mA, OC = 16mA output current.
Symbol with backslash or bar is low active.

FUNCTIONAL DESCRIPTION

MULTIPLIER

A 16x16 with 32-bit result multiplier is included. Efficient FIR operation can be realized through MB and MPA instructions. Since multiplication is an important operation in DSPalgorithm, flexible multiplication is important in DSP implementation. The MX96037 supports complex multiplication (MB) for communication application. General multiplication operations may use MPA and MX.

LOOP

Repeat or loop instruction is important in DSP operation. The MX96037 supports this function by implementing many instructions which are implictly repeated with the number stored in the RCR register. Examples are TBR, MPA, etc. Loop up to 8 instructions with specified number of times (can be variable) is also implemented with hardware. Again, flexible usage format is supported which makes the instruction more useful.

MODULAR ADDRESSING

Modular addressing is by modular operation at the output of ARAU. To use modular addressing user must first store non-zero number m which is stored to the MODR register. With this in effect, memory space beginning from $k \cdot 2^n$ to $k \cdot 2^n+m$, where k is an integer greater than or equal to zero and 2^n is a power-of-two integer greater than m, will form a circular memory space. Whenever boundary location, 0 or m, is addressed, the next AR content will be set/reset to m/0, independent of the instruction specification. Set MODR to 0 will deactivate modular addressing. For example, if MODR is set to 23, circular memory spaces will start from $32 \cdot k$ to $32 \cdot k+23$. Any instruction can be indirectly addressed to 55, assuming that using AR1, with increasing operation, will make the next AR1 content to be reset to 32. Likewise, if AR1 content is in decreasing operation and the content of AR1 is set to 0, then the next value of AR1 will be reset to 23. If normal addressing mode is desired, simply output a 0 into the MODR registers. This instruction can help construct data RAM into circular buffer or delay line, thereby eliminating the need of physical data movement in the buffer or delay. However, the pointer need to be kept in the data RAM for easy access to the head/tail of this buffer/delay line.

AUXILIARY REGISTERS

Eight 16 bits auxiliary registers are allocated together with a 16-bit adder/subtractor. The results of adder/subtractor always go through a modulator to get modular addressing before being stored to the auxiliary registers.
The process provides an independent processor to do address calculation and update in parallel with main data path which performs the instruction execution. Of course, AR registers can also be used as temporary registers and as another unsigned adder/subtractor. AR register modification of $\pm$(0,1,2,AR0) on the fly is also included.

STACK

Hardware contains 16 deep dedicated stack memories, which support deep hierarchy code. Stack manipulation is transparent to firmware.

RAM/ROM MAP

The MX96037 provides 64k words address space for both RAM and ROM.18k words ROM and 2k words RAM are on- chip. The other memory spaces are for external use. The internal 18k words ROM can be disabled by pulling EROM pin high, then external memory will be addressed for the first 18k words spaces. The MX96037 also provides 32 on-chip IO port for DSP use. These IO ports can be accessed by IN, OUT or specific instructions (see register definition).

HOLD

Hardware hold is supported through pins $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$. When $\overline{\text{HOLD}}$ is activated, the MX96037 will enter hold state after the present instruction cycle is completed(instructions inside Loop and inherent repeat instruction cycles is considered one instruction cycle). At hold state, the MX96037 will release address and data bus to high impedence, stop executing instruction and output $\overline{\text{HOLDA}}$. After $\overline{\text{HOLD}}$ is invalid the MX96037 will bring $\overline{\text{HOLDA}}$ to high and resume normal operation.

HOST INTERFACE

UPMODX=0 will enable µp command interface hardware and firmware to support external µp operations.

If UPMODX=0, CMDR, a 16-bit bidirection register, can be accessed by HOST via PHDB(7:0) pins in two accesses selected by PHILO\ pin. DSP accesses CMDR by 16-bit width. When HOST writes to high byte of this register, CRDY bit in register 7 (bit 6) will be set. When DSP reads CMDR, CRDY bit will be reset.
PACKB\ signal is defaultly set to high. When DSP writes to CMDR, PACKB\ is reset(active low). The host read will set this PACKB\ pin to high.
External µp may use this parallel interface to set DAM_BIOS command and read responses from this port.

SYSTEM TICK

System tick always exists. It periodically interrupts DSP every 1 ms in normal mode or 32 ms in power down mode unles system tick interrup is masked (IMR:STMRM=1). Standard timer system may be implemented by this tick.

POWER MANAGEMENT

There are several ways to do power management. Users can program, PWDN or/and SWHOLD or assert HOLD\.

1. if SWHOD bit is set or hardware HOLD\ signal is activated, DSP operations and Codec clock will be halted.
Any interrupt source at SWHOLD mode will reset SWHOLD then DSP will resume normal operation. Interrupt can't affect HOLD\.

2. When PWDN is set, DSP will run at 32.768 kHz and hi-crystal oscillation circuit is de-activated until it is reset to zero.
Normally, It takes 62 mini second to re-activate crystal oscillation circuit and status bit LSRUN indicates the speed of DSP of current operation.

INTERNAL/EXTERNAL MEMORY CONSIDERATIONS

1. DSP internal RAM is always selected, if external RAM is overlapped, the external portion is invisible by DSP.

2. DSP internal ROM may be disabled by active EROM pin. For initial program development, EROM\ may be active to use full range of external ROM.

   For sake of smarter ROM planning , DSP could write a test of external ROM presence, if external rom exists, just jump to the external directly.

   There are many possible configurations to be considered such as building DSP fast routine in internal ROM (just like IBM PC BIOS) or basic control system inside while leaving flexibility outside using shower memory.

REGISTER DESCRIPTIONS

NAME	BIT	CTLR	IO ADDRESS	RELATED INSTRUCTIONS	DESCRIPTIONS
optr	16	o	0	IN/OUT	output register
iptr	8	o	1	IN	input port register
bior	16	o	2	IN/OUT	bidirectional io register
svr	4		3	IN/OUT/SFR/SFL	shifter count (scr) and sign
imr	4		4	IN/OUT	interrupt mask register
isr	3		5	IN	interrupt status register
ctrl	15		7		control register
wstr	8		8	IN/OUT	wait state control register
mmacr	16	o	9	IN/OUT	dram access control register
mmapl	15	o	10	IN/OUT	dram access pointer low
mmaph	16	o	11	IN/OUT	dram access pointer high
rcr	7		12	IN/RPT/TBR/MPA/LUP	repeat counter
modr	7		13	IN/MOD	modulo register
xr	16		14	IN/LX/LXA/LXM/LXS multiply instructions	one of multiplier operands
sp	4		15	PSH/POP/IN/PSHH/POPH TBR, MPA,PSHL,POPL	stack pointer register
cdrr0	16	o	16	IN	codec 0 receive buffer
cdxr0	16	o	17	OUT	codec 0 transmit buffer
pregl	16		18		product register low word
pregh	16		19		product register high word

REGISTER DESCRIPTIONS *(Continued)*

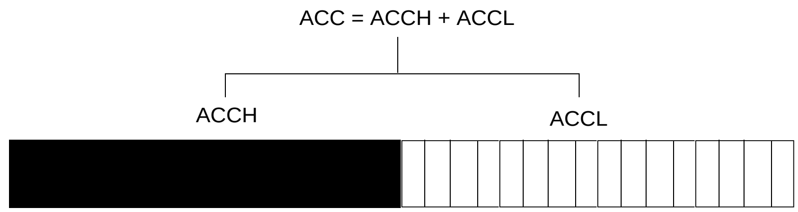
NAME	BIT	CTLR	IO	RELATED INSTRUCTIONS	DESCRIPTIONS
tstr	8	o	20	IN/OUT	test and codec control register
acch	16		–	(many instr.)	DSP accumulator, the basic ALU
accl	16		–	SAL/ADL/SBL	
ar0-7	16x8		–	LAR/MAR/SAR	for indirect memory access basically; also used in macro instructions
accx	32		–	SBL, ADL, SFL SFR,multiply	acch+accl=accx
p	32		–	multiply instructions PAC/APAC	product register
ssr	16			SSS/OUT/BS/BZ INTM : EINT/DINT TB : BIT OVM : ROVM/SOVM ARP : MAR	status register
pc	16		–	CALL, CALA, TRAP, BS, BZ BACC, RET, RETI, interrupt, hardware reset	program counter

TABLE OF IO MAPPED REGISTERS

Register	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Access
OPTR:(00)	OPT15	OPT14	OPT13	OPT12	OPT11	OPT10	OPT9	OPT8	OPT7	OPT6	OPT5	OPT4	OPT3	OPT2	OPT1	OPT0	RW
IPTR:(01)									IPT7	IPT6	IPT5	IPT4	IPT3	IPT2	IPT1	IPT0	RO
BIOR:(02)	BIOR15	BIOR14	BIOR13	BIOR12	BIOR11	BIOR10	BIOR9	BIOR8	BIOR7	BIOR6	BIOR5	BIOR4	BIOR3	BIOR2	BIOR1	BIOR0	RW
SVR:(03)													SCR3 ~ SCR0				RW
IMR:(04)													SSM	1 STMRM	1 CODCM	1 INTIM	RW
ISR:(05)														0 STMRS	0 CODCS	0 INTIS	RO
CTLR:(07)		OPT18	OPT17	OPT16	PWDN	SWHOLD	FAST1	FAST0	HMOD	CMDRDY	LSRUNS	SS	HSSRC	SNSEL	UPMODX	C6KSEL	RW
WSTR:(08)									MMSIZE		SRAMWAIT		MMWAIT		ROMWAIT		RW
MMAC:(09)	MMCNT (Mass Memory move Count, 6 bits)						IRA (Internal RAM Address, bank one, 10 bits)										RW
MMAPL:(10)																	RW
MMAPH:(11)	0 TOIRAM																RW
RCR:(12)																	RO
MODR:(13)	^	^	^	^	^	^	^	^	^								RO
XR:(14)																	RO
SPR:(15)																	RO
CDRR:(16)																	RO
CDXR:(17)																	WO
PREGL:(18)																	WO
PREGH:(19)																	WO
TSTR:(20)																	RO

*NOTE : Register 6 is revised for future use.

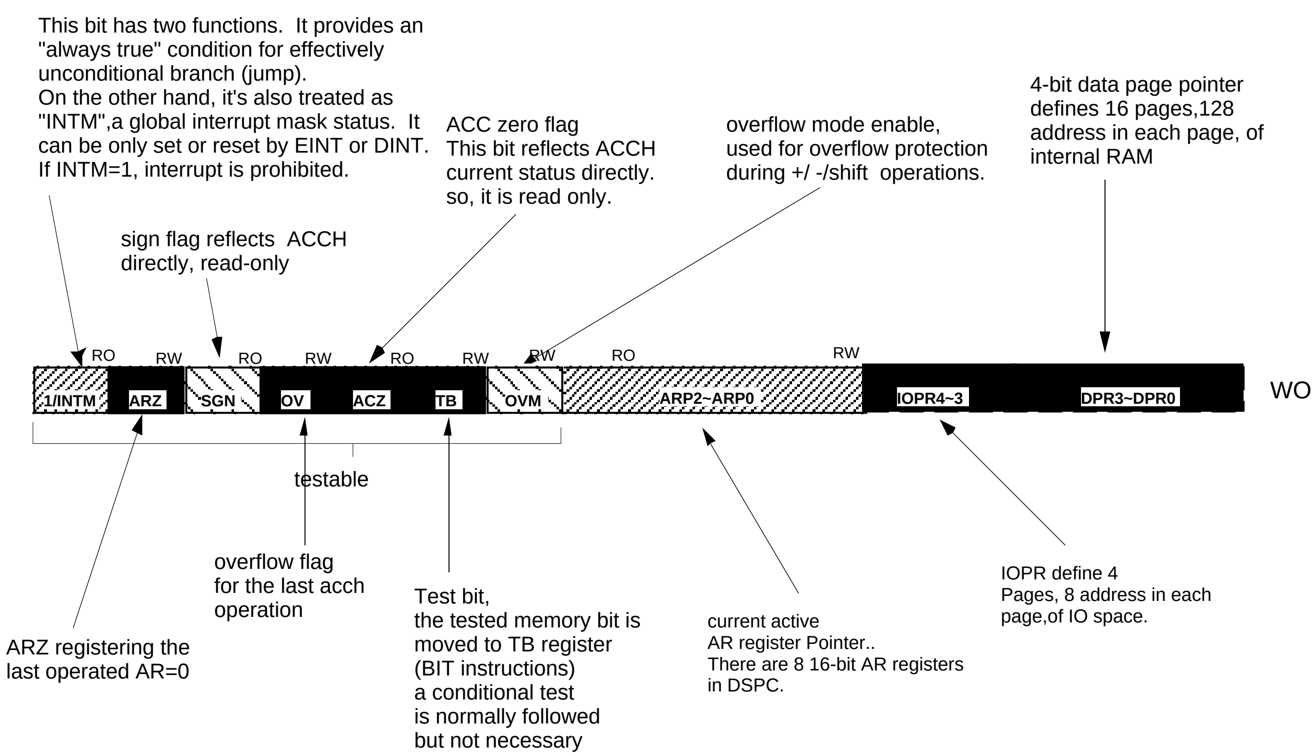
1. ACC:ACCUMULATOR



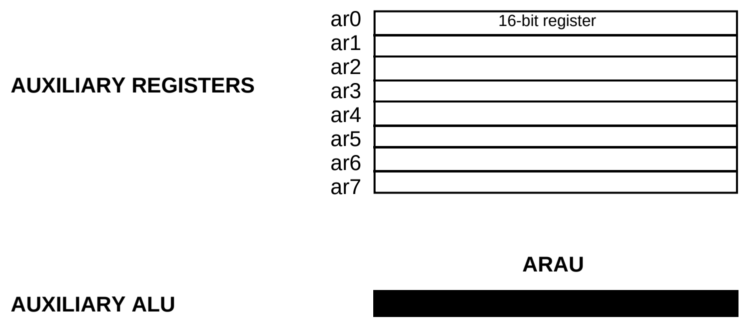
- acch+accl=acc
- Logic ALU operation is 16 bits and executed on acch. (ACCL is not affected)
- ADL/SBL is 32-bit operation. (SVR:SNSEL determine sige-extended)
- Lac will put ACCL to 0

SSR : STATUS REGISTER

ssr includes 8 testable status/register bits (ssr:15~8), and 3 arp bits, 2 IOP bits, 4 DP bits
SGN and ACZ reflect status of ACCH. (can not be saved)



AR (AUXILIARY REGISTER) AND ARAU (AUXILIARY ALU)



- 16x8 AR registers provide powerful indirect memory access.
- Modulo addressing (modulo memory indexing) provides easy implementation of ring buffer or delay line. See modulo register (MODR) for more details.
- ARAU provides +/ - (0, 1, 2, AR0) post execution after each addressing of ARs.
- ARAU works in parallel with main ALU.
- ARs may be also used as scratch registers.

PC AND PROGRAM FLOW CONTROL

Program flow is affected by:

1. BS/BZ (branch-if-set/branch-if-zero) conditional branch.
2. BACC - branch indirectly by ACCH.
3. CALA - call indirectly by ACCH, return address is pushed.
4. CALL - call subroutine, see 'Addressing Modes, Misc. Addressing mode'
5. TRAP - Trapped to call fixed hex 000C address.
6. Power-on reset and interrupt see 'interrupt Operations'

ADDRESSING MODES

IMMEDIATE CONSTANT

Immediate constant is coded directly in opcode.

DIRECT MEMORY ADDRESSING

DPR and IOPR are used to completely specify addressing spaces. 4 bits in DPR combined with 7-bits coded in opcode, make direct memory address. (direct memory addressing ONLY FOR INTERNAL 2K WORDS RAM)

INDIRECT ADDRESSING

The memory address may be pointed by ARs. ARs also has post-addressing execution which provides powerful increment(s)/decrement(s) and modulo indexing.

It takes only 7 bits to code all these into one opcode to enable program size compact. See AR, ARAU and MODR for more details.

MISCELLANEOUS ADDRESSING MODE

CALL--Call subroutine at the second word of call instruction.
CALA--ACCH indirect call, ACCH=called address
BACC-- ACCH indirect branch, ACCH=branch address
TRAP-- Always call to hex 000C address

MACRO OPERATIONS

- LUP - repeat instruction-block (max 8 instructions)
  - repeat count = RCR +1
  - Each time PC hits loop boundary (starting address +LC), loop count is increased by 1
- MPA - Vectored inner product, see 'P, X, Y Register and Multiplier Operations.

INTERRUPT : OPERATIONS

The Interrupt source, vectoring address and priority are as follows:

NAME	VECTORED ADDRESS	DESCRIPTIONS
RST\	0000	Power-on reset (top priority)
NMI\	0002	NMI\non.maskable interrupt, edge-triggered (high to low)
SS	0004	Single-Step, Single step interrupt is for debugging purpose. If set, MX96037 will be interrupted after every instruction cycle (instructions inside LOOP and inherent repeat instruction cycles is considered as one instruction cycle). User can put debugging
service		as the interrupt service routine.
INT1\	0006	INT1\ pin interrupt, edge trigged
CODEC	0008	Triggered when Codec registers get/send 16 bit data (see Timing diagram)
STMR	000A	Triggered every 1 msec
TRAP	000C	Triggered when executes TRAP

Interrupt Process: (Execute by hardware)

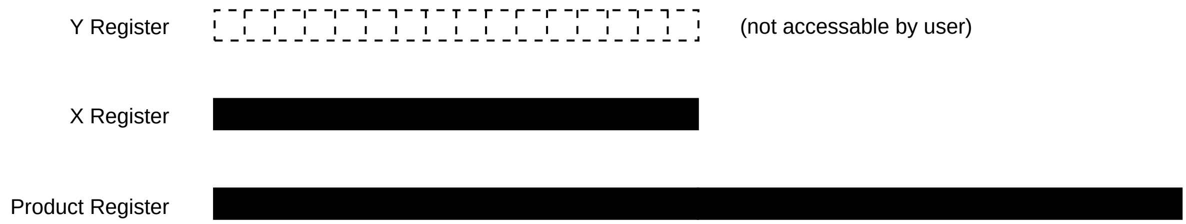
1. Release related ISR pending flag
2. Push SSR on to stack
3. Push return-address on to stack
4. Disable global interrupt (same to excuting DINT instruction)
5. If it is in software hold state (see WSTR register and power management), reset SWHOLD → 0, and come out of software hold state.

Issues of RETI instruction: (Execute by hardware)

1. POP return address to PC
2. POP SSR

Note that ACC normally need to be saved. All other registers should also be carefully maintained when doing an in-and-out interrupt.

P, X, Y REGISTER AND MULTIPLIER OPERATIONS



Multiplier operand is fed from X and Y registers (see Architecture). X is visible by instruction, Y is supplied from various source (such as constant, memory, acch).

The result of multiplication is put in p register. Accumulation is done through pipelining.

Note that the result of multiplication is always shift-left one bit before being put in register(do not use ox 8000 multiply which will cause overflow).

BASIC MULTIPLY

MX/MXP (x)*(dma) → p
MXK/MXL (x)*constant → p

ACCUMULATE PREVIOUS PRODUCT AND MULTIPLY

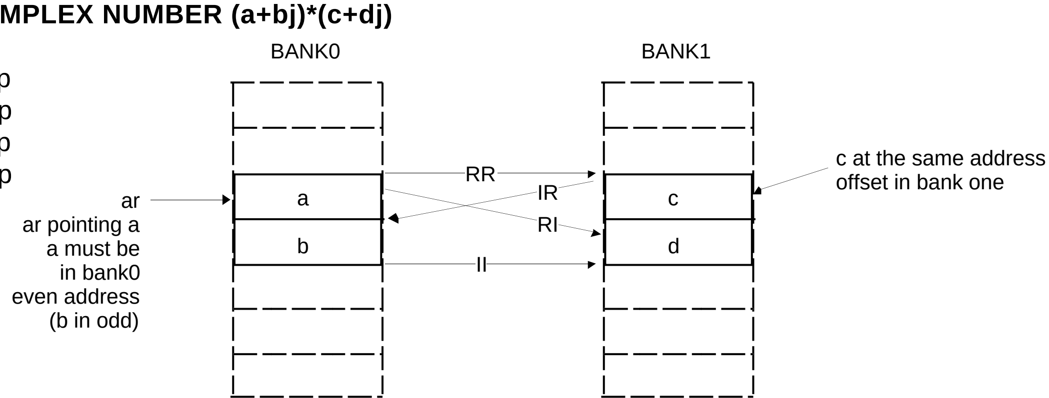
MXA/MXAP (accx) + (p) → accx, (x) * (dma) → p

SUBTRACT PREVIOUS PRODUCT AND MULTIPLY

MXS/MXSP (accx) - (p) → accx, (x) * (dma) → p

MULTIPLY BANKED COMPLEX NUMBER (a+bj)*(c+dj)

MB	RR, *	a*c → p
MB	RI, *	a*d → p
MB	IR, *	b*c → p
MB	II, *	b*d → p



Note : When working with repeat counter (rcr), loop instruction and ARAU operation, array of complex number may be calculated efficiently.

MULTIPLY BANK AND ACCUMULATE

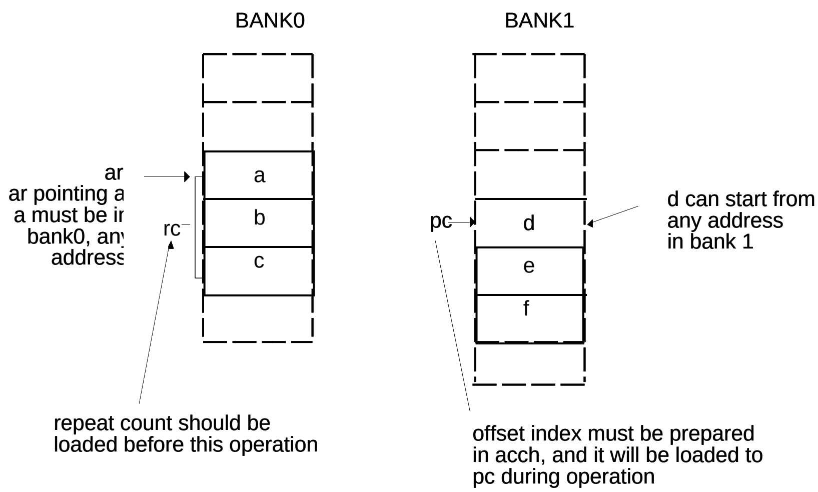
MBA	RR, *	(acch) + (p) → acch, a*c → p
MBA	RI, *	(acch) + (p) → acch, a*d → p
MBA	IR, *	(acch) + (p) → acch, b*c → p
MBA	II, *	(acch) + (p) → acch, b*d → p

MULTIPLY BANK AND SUBTRACT

MBS	RR, *	(acch) - (p) → acch, a*c → p
MBS	RI, *	(acch) - (p) → acch, a*d → p
MBS	IR, *	(acch) - (p) → acch, b*c → p
MBS	II, *	(acch) - (p) → acch, b*d → p

VECTORS INNER PRODUCT

X = [a b c]
Y = [d e f] → X*Y=a*d+b*e+c*f



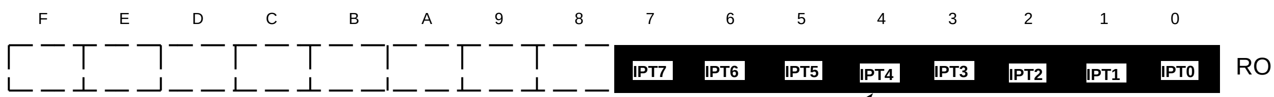
MPA *, NAR
[(accx) +(p)→ p, (ar)*(pc)→ p, (pc) +1 →pc] repeat rc+1 times.

IO REGISTERS

OPTR : Output Register (mapped to IO register 00)

Output Register (OPTR:15 ~ OPTR:0)
16 bit, connect to OPT15~OPT0 pins.
positive Logic, '1' will output 5 Volt on output pin. ('0' for 0 V)
All these bits can memory output value (read by 'IN'
instruction) to eliminate shadow memory requirement.

IPT : Input Port Register (mapped IO address 01)



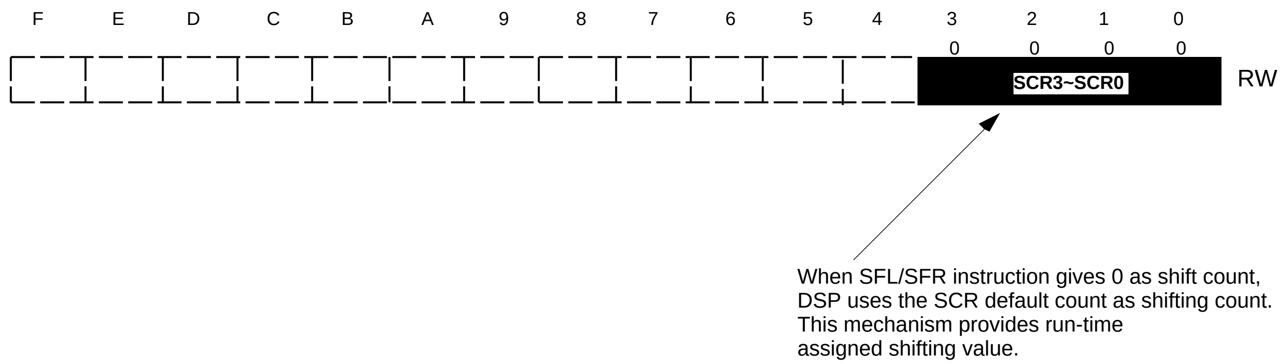
Input Port
Positive Logic, 5 Volt input will read '1' (0V for '0')
IPT:7~IPT:0 connect IPT7~IPT0 pins.
IPT3~IPT0 are internally pull high

BIOR/CMDR : BI-DIRECTION IO REGISTER IN NON-UP MODE OR COMMAND REGISTER IN UP MODE (mapped to IO register 02)

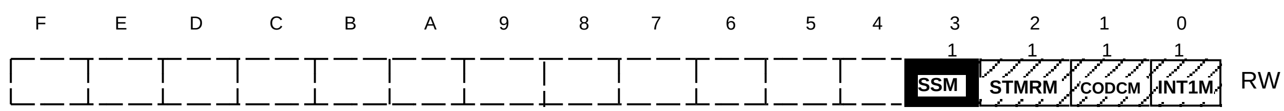
as UPMODX=1, used for bidirectional io register.
Programable bidirectional IO.
BIOR15~BIOR8 control I/O direction of BIOR7~BIOR0, respectively (bit 8 control bit 0)
BIOR7~BIOR0 connect to BIO7~BIO0 pins, respectively.
UPMODX=0, used for 16-bit parallel interface command register.

SVR : Shift Variable Register (mapped to IO register 03)

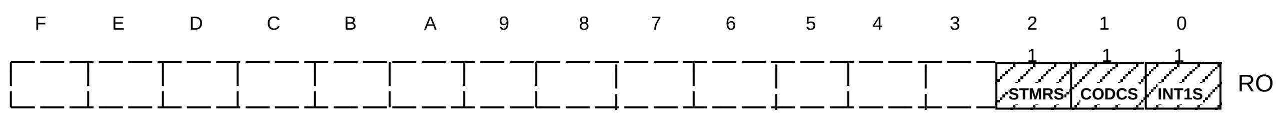
SVR includes Shift-Count Register (SCR)



IMR : Interrupt Mask Register (mapped to IO register 04)



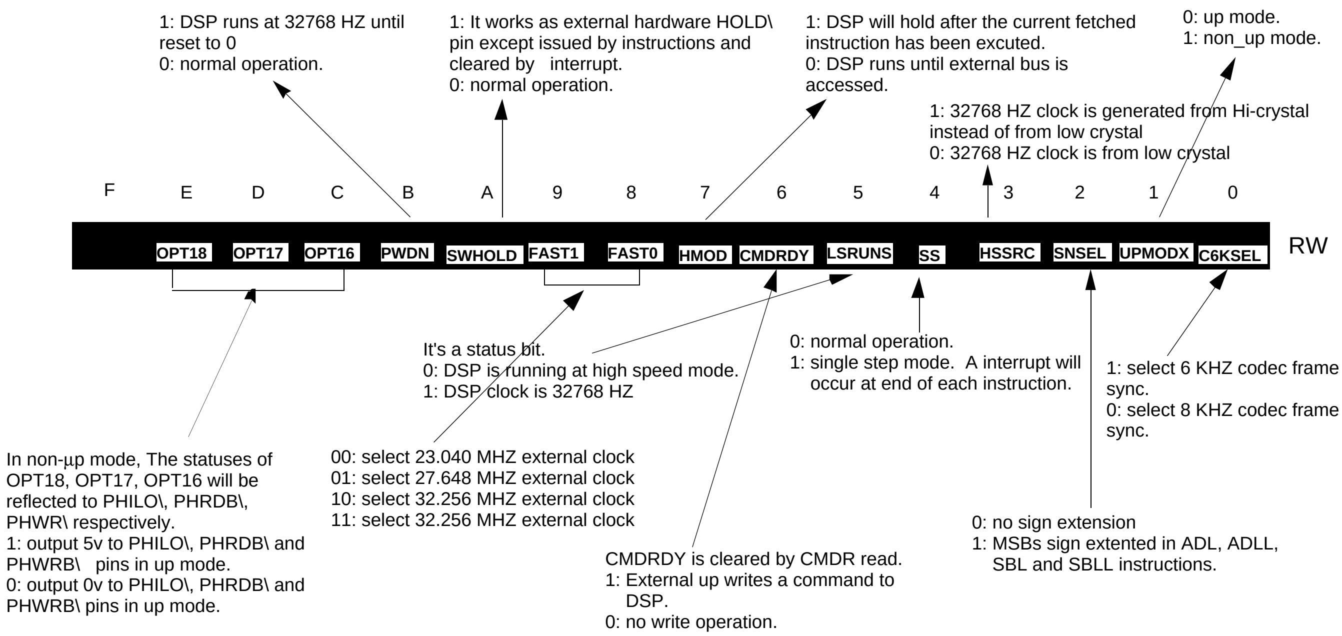
ISR : Interrupt Status Register (mapped to IO register 05)



INT1M - INT1 \ Interrupt Mask 1
STMRM - System tick Timer interrupt Mask
CODCM - Codec Interrupt Mask
SSM - Single Step Interrupt Mask

Note 1: Codec Tx/Rx use this same mask.
This is because the 2 events are synchronized and always happen at the same time. Programmers should take care of these 2 events (if necessary) in this interrupt.
Note 2: ISR:2~0 will reflect interrupt pending status on IMR:2~0.
Note that Single-Step (CTLR:SS, register 07) is directly controlled by the program; no status exists.
Note 3: Read ISR will read and clear all pending flags.

CTLR : CONTROL REGISTER (MAPPED TO IO REGISTER 07)



WAIT STATE:

ROM/RAM Timing Requirement = (3+2W)C- 20ns
W=0/1/2/3 (wait state number on IO/RAMWAIT/ROMWAIT field)
C=21.7/18.08/15.5ns (internal clock timing for external crystal 23.040/27.648/32.256 MHz)

	WAIT-STATE NUMBER	**0**	**1**	**2**	**3**
ROM/RAM	C=15.5 C=18.08 C=21.7	26.5 34.2 45.1	57.5 70.8 88.5	88.5 107.4 133.6	119.5 144.0 179
DRAM		-	-	-	70-120ns

NOTE: The MX96037 may have 23.040/27.648/32.256 MHz crystal options. This clock is doubled internally. Instructions will use 3 clocks as an instruction cycle (SAH/SAHP, SAL/SALP, SSS/SSP extend 1 internal clock).

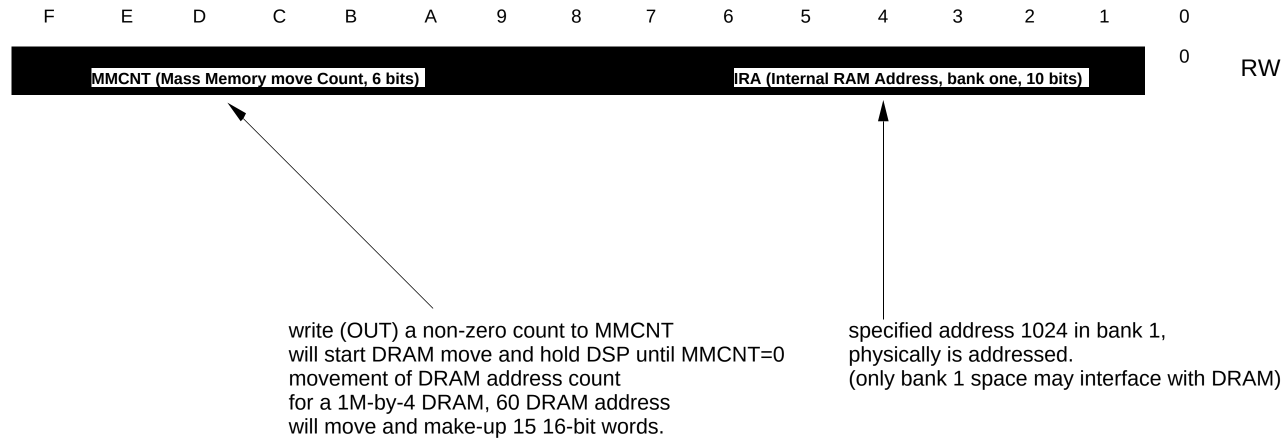
WSTR: WAIT STATE, SINGLE-STEP AND DRAM SIZE (MAPPED TO IO REGISTER 08)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
								MMSIZE		SRAMWAIT		MMWAIT		ROMWAIT		RW

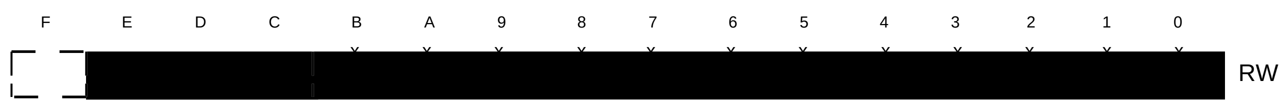
Mass Memory Size (ARAM)
Select the external DRAM bit-size
0 0 -- 1 bit
0 1 -- 4 bits
1 0 -- 8 bits
1 1 -- 16 bits

DRAM CONTROLLER : MMACR, MMAPH AND MMAPL

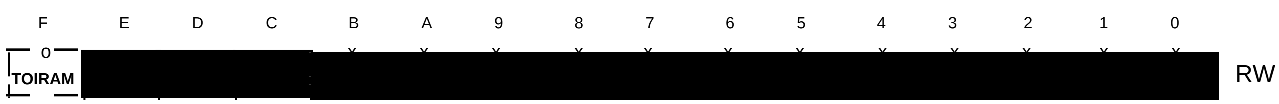
MMACR : Mass Memory Access Control register (mapped to IO register 9)



MMAPL : Mass Memory Access Pointer Low register (mapped to IO regsiter 10)

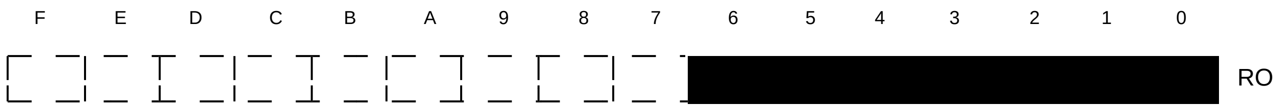


MMAPH : Mass Memory Access Pointer High register (mapped to IO register 11)



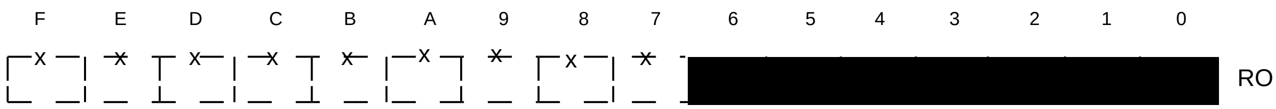
TOIRAM=1, DRAM $\rightarrow$ Internal RAM
TOIRAM=0, Internal RAM $\rightarrow$ DRAM
MMAPH+MMAPL make-up 30 bits, 1G addressing space.
This space may have max 1G words DRAM capacity.

RCR : Repeat Counter Register (mapped to IO register 12)



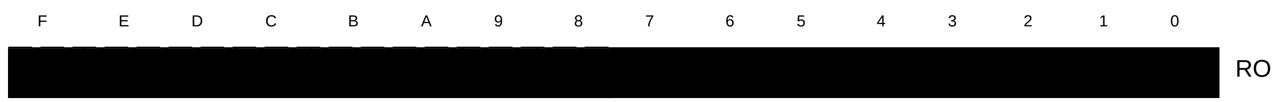
- RCR provides repeat count on TBR, LUP, and MPA types macro instructions.
- RCR must be prepared before macro instructions are being executed. (RPT instruction)
- Repeat time is RCR+1

MODR : MODULAR REGISTER (MAPPED TO IO REGISTER 13)



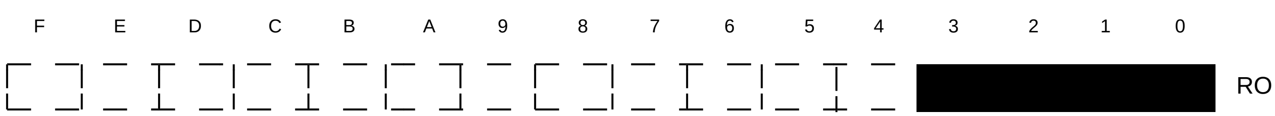
As MODR=M $\rightarrow$ 0, 1, 2, .., M-1, M modulo mechanism will be enforced (note: bounded by M --- not M-1)
Modular addressing is always performed at the output of ARAU.
As MODR=0, modulo addressing is disabled.
MOD type instructions are used to load MODR; use IN instruction to read MODR.

XR : X REGISTER (MAPPED TO IO REGISTER 14)



- One of multiplier inputs
- Can be accessed by IN/LX/LXM/LXS/LXA/LXAK etc... instructions.

SPR : Stack Pointer Register (mapped to IO register 15)



- 16-level stack provides nestable interrupt and controller level nested call capabilities.
- SPR is pointing to 'next-available' word, and initialized to 0.
- As SPR is over address 15, it wraps around to 0. As SPR=0, POP will also wrap SPR to 15.
- SPR can only be read by IN instructions; no write capability.

CDRR : Codec Data Receive Register (Mapped to IO register 16)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
																RO

CDXR : Codec Data Transmit Register (Mapped to IO register 17)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
																WO

1. Two Codec events from the above registers are always synchronized, so there is only one Codec interrupt for them.
2. These codecs are in 16-bit mechanism; however, 8-bit Codec is also applicable.
   In 8-bit case, to tx, the data byte to be transmitted must be in bit15~bit8. The received data byte is at bit15~bit8 as read from receive register.
3. MSB (Most-Significant-Bit) is shifted first.
4. Codec registers has shadow registers as buffer.

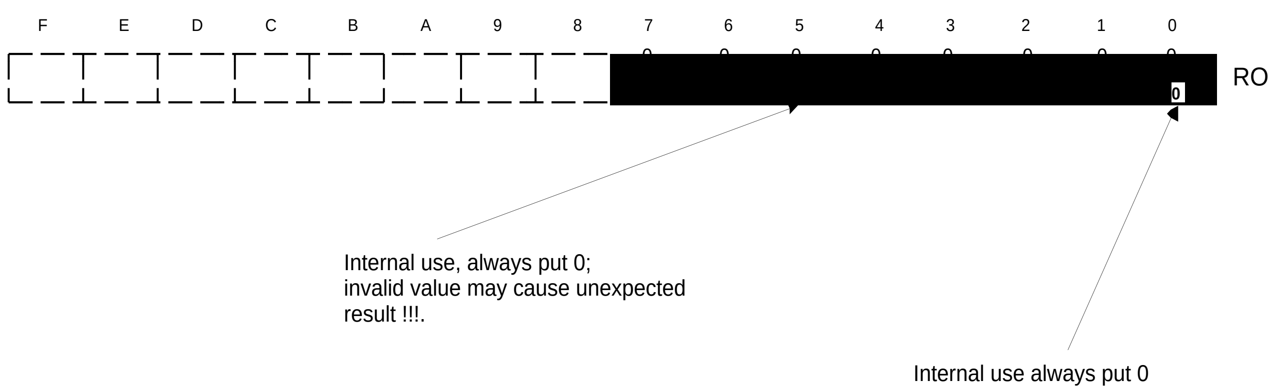
PREGL : Product Register Low Word (Mapped to IO register 18)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
																WO

PREGH : Product Register High Word (Mapped to IO register 19)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
																WO

TSTR: Test Register Mapped to Register (mapped to register 20)



INSTRUCTION SET SUMMARY

ABBREVIATIONS

a	:	AR pointer
ar	:	AR
acc	:	accumulator
c	:	short constant
d	:	data memory address
dp	:	data page pointer
i	:	Addressing mode select bit
k	:	odd/even address select
l	:	loop counter
L	:	constant for shift left
mr	:	modulo register
mb0	:	internal memory bank 0
mb1	:	internal memory bank 1
o	:	io page pointer
p	:	product register
pa	:	port address
pc	:	program counter
R	:	constant for shift right
rc	:	repeat counter
s	:	shift right sign extention select bit
sp	:	stack pointer
ss	:	status register
sv	:	shift register
x	:	don't care
xr	:	x register
y	:	AR arithmetic operation select

		Mnemonic and Description	Words & cycles	16-bit opcode MSB			LSB
abs	;	absolute value of accumulator	1,1	1001	1000	0xxx	xxxx
adh	;	add to high acc	1.1	0000	0000	iddd	dddd
adhk	;	add to high acc. short immediate	1,1	0000	0001	0ccc	cccc
adhl	;	add to high acc. immediate	2,2	1000	0000	0xxx	xxxx
adl	;	add to low acc	1,1	0000	0010	iddd	dddd
adlk	;	add to low acc. short immediate	1,1	0000	0011	0xxx	xxxx
adll	;	add to low acc. immediate	2,2	1000	0001	0xxx	xxxx
and	;	and with high acc	1,1	0000	1010	iddd	dddd
andk	;	and short immediate with high acc	1,1	0000	1011	0ccc	cccc
andl	;	and immediate with high acc	2,2	1000	0101	0xxx	xxxx
apac	;	add p reggister to acc	1,1	1001	0010	0xxx	xxxx
bacc	;	branch to address specified by acc	1,2	1111	1010	0xxx	xxxx
bit	;	test bit	1,1	0110	bbbb	iddd	dddd

		Mnemonic and Description	Words & cycles	16-bit opcode MSB			LSB
bs	;	branch immediate if bit set	2,3	1101	1bbb	0xxx	xxxx
bz	;	branch immediate if bit reset	2,3	1101	0bbb	0xxx	xxxx
cala	;	call subroutine indirect specified by acc	1,2	1100	0000	0xxx	xxxx
call	;	call subroutine	2,3	1111	1100	0000	0000
dint	;	disable interrupt	1,1	1111	0000	0xxx	xxxx
eint	;	enable interrupt	1,1	1111	0001	0xxx	xxxx
in	;	input data from port	1,1	1010	ppp0	iddd	dddd
lac	;	load acc	1,1	0000	1110	iddd	dddd
lack	;	load acc. short immediate	1,1	0000	1111	0ccc	cccc
lacl	;	load acc. immediate	2,2	1000	0111	0xxx	xxxx
lar	;	load auxiliary register	1,1	0111	aaa0	iddd	dddd
lark	;	load auxiliary register short immediate	1,1	0111	aaa1	0ccc	cccc
larl	;	load auxiliary register immediate	2,2	1000	1000	0aaa	0000
ldp	;	load data page register	1,1	0001	0100	iddd	dddd
ldpk	;	load short immediate to data page register	1,1	0001	0101	0xxx	cccc
lip	;	load io page register	1,1	0001	0010	iddd	dddd
lipk	;	load io page register with short immediate	1,1	0001	0011	0xxo	oxxx
lup	;	loop instruction	1,1	0101	0100	iddd	dddd
lupk	;	load rc with 7-bit constant and enable loop operation	1,1	0101	0101	0ccc	cccc
lx	;	load x register	1,1	0010	0000	iddd	dddd
lxk	;	load short immediate to x register	1,1	0010	0001	0ccc	cccc
lxl	;	load immediate to x register	2,2	1000	1010	0xxx	xxxx
lxa	;	load x register and accumulate previous product	1,1	0011	1100	iddd	dddd
lxak	;	load short immediate to x registeri and accumulate previous product	1,1	0011	1101	0ccc	cccc
lxal	;	load immediate to x register and accumulate previous product	2,2	1000	1011	0xxx	xxxx
lxm	;	load x register and store p register to acc	1,1	0011	1110	iddd	dddd
lxmk	;	load short immediate to x register and store p register to acc	1,1	0011	1111	0ccc	cccc
lxml	;	load immediate to x register and store p register to acc	2,2	1000	1100	0xxx	xxxx
lxs	;	load x register and subtract previous product	1,1	0011	0100	iddd	dddd
lxsk	;	load short immediate to x register and subtract previous product	1,1	0011	0101	0ccc	cccc
lxsl	;	load immediate to x register and subtract previous product	2,2	1000	1101	0xxx	xxxx
mba	;	multiply and accumulate previous product	1,1	0011	1000	00kk	0yyy
mbs	;	multiply and subtract previous product	1,1	0011	0000	00kk	0yyy
mar	;	modify auxiliary register	1,1	1111	0110	iddd	dddd

		Mnemonic and Description	Words & cycles	16-bit opcode MSB			LSB
mod	;	load modulo register	1,1	0001	0110	iddd	dddd
modk	;	load modulo register short immediate	1,1	0001	0111	0ccc	cccc
mpa	;	array multiplication	1,3+rc	1100	1110	iddd	dddd
mpc	;	ram_bank_0 multiply ram_bank_1	1,1	0010	0100	00kk	0yyy
mx	;	(x) multiply (dma)	1,1	0010	0010	iddd	dddd
mxk	;	(x) multiply (7-bit constant)	1,1	0010	0011	0ccc	cccc
mxl	;	(x) multiply (16-bit constant)	2,2	1000	1110	0xxx	xxxx
mxa	;	(x) multiply (dma) and accumulate previous product	1,1	0011	1010	iddd	dddd
mxak	;	(x) multiply (7-bit constant) and accumulate previous product	1,1	0011	1011	0ccc	cccc
mxs	;	(x) multiply (dma) and subtract previous product	1,1	0011	0010	iddd	dddd
mxsk	;	(x) multiply (7-bit constant) and subtract previous product	1,1	0011	0011	0ccc	cccc
nop	;	no operation	1,1	1111	1111	1111	1111
or	;	or with high acc	1,1	0000	1000	iddd	dddd
ork	;	or short immediate with high acc	1,1	0000	1001	0ccc	cccc
orl	;	or immediate with high acc	2,2	1000	0100	0xxx	xxxx
out	;	output data to port	1,1	0100	ppp0	iddd	dddd
outk	;	output short immediate to port	1,1	0100	ppp1	0ccc	cccc
outl	;	output immediate to port	2,2	1000	1111	0ppp	0000
pac	;	load acc. with p register	1,1	1001	0100	iddd	dddd
poph	;	pop top of stack to high accumulator	1,1	1001	1010	iddd	dddd
popl	;	pop top of stack to low accumulator	1,1	1001	1011	iddd	dddd
pop	;	pop top of stack to data memory	1,1	1011	0101	iddd	dddd
pshh	;	push high accumulator onto stack	1,1	1100	1000	iddd	dddd
pshl	;	push low accumulator onto stack	1,1	1100	1001	iddd	dddd
psh	;	push data memory value onto stack	1,1	1100	1010	iddd	dddd
rovm	;	reset overflow mode	1,1	1111	0100	0xxx	xxxx
rxf	;	reset external flag	1,1	1111	0010	0xxx	xxxx
ret	;	return from subroutine	1,2	1111	1000	0xxx	xxxx
reti	;	return from interrupt	1,2	1111	1001	0xxx	xxxx
rpt	;	load repeat counter	1,1	0001	0000	iddd	dddd
rptk	;	load rc with 7-bit constant	1,1	0001	0001	0ccc	cccc
sal	;	store low acc	1,1	1011	0001	iddd	dddd
sar	;	store auxiliary register	1,1	1011	1aaa	iddd	dddd
sfl	;	shift acc left	1,1	1001	1100	0000	LLLL
sfr	;	shift acc right	1,1	1001	1110	000s	RRRR
sovm	;	set overflow mode	1,1	1111	0101	0xxx	xxxx
spac	;	subtract p register from acc	1,1	1001	0110	0xxx	xxxx
sqra	;	square and accumulate previous product	1,3+rc	1111	1011	iddd	dddd
sip	;	store iopage register	1,1	1011	0010	iddd	dddd
sss	;	store ss register	1,1	1011	0011	iddd	dddd
sdp	;	store datapage register	1,1	1011	0100	iddd	dddd
sbh	;	subtract from high acc	1,1	0000	0100	iddd	dddd
sbhk	;	subtract short immediate from high acc	1,1	0000	0101	0ccc	cccc
sbhl	;	subtract immediate from high acc	2,2	1000	0010	0xxx	xxxx

		Mnemonic and Description	Words & cycles	16-bit opcode MSB			LSB
sbl	;	subtract from low acc	2, 2	0000	0110	iddd	dddd
sblk	;	subtract short immediate from low acc	1,1	0000	0111	0ccc	cccc
sbll	;	subtract immediate from low acc	1,1	1000	0011	0xxx	xxxx
sxf	;	set external flag	1,1	1111	0011	iddd	dddd
tbr	;	table read	1,3+rc	1100	1100	iddd	dddd
trap	;	software interrupt	1,2	1100	0010	0xxx	xxxx
xor	;	xor with high acc	1,1	0000	1100	iddd	dddd
xork	;	xor short immediate with high acc	1,1	0000	1101	0ccc	cccc
xorl	;	xor immediate with high acc	2,2	1000	0110	0xxx	xxxx

abs **absolute value of accumulator.**

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

SYNTAX: ABS

EXECUTION:
(pc) + 1 → pc
|acc(31:16)| → (acc (31:16))

WORDS: 1

CYCLES: 1

adh **add to high acc.**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	1	see note 1						

SYNTAX:
adh dma7
adh *(,nar)

EXECUTION:
(pc) + 1 → pc
(acc(31:16))+(dma) → (acc (31:16))

WORDS: 1

CYCLES: 1(DI) 2(DE)

Adhk **add to high acc. Short immediate.**

BIT:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0	7-bit constant						

SYNTAX: adhk cnst 7

EXECUTION:
(pc) + 1 → pc
(acc(31:16)) + (7-bit constant) → (acc (31:16))

WORDS: 1

CYCLES: 1

adhl **add to high acc. Immediate.**

BIT:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0							
16-bit constant															

SYNTAX: adhl cnst16

EXECUTION: (pc) + 2 → pc
(acc(31:16))+(16-bit constant) → (acc (31:16))

WORDS: 2

CYCLES: 2

adl **add to low acc.**

Direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	data memory address						

Indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	see note 1						

SYNTAX: adl dma7
adl *(,nar)

EXECUTION: (pc) + 1 → pc
(acc)+(dma with optional MSBs sign extension) → (acc)

WORDS: 1

CYCLES: 1(DI) 2(DE)

NOTE: Option is controlled by CTRL: SNSEL bit

adlk **add to low acc. Short immediate.**

BIT:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	1	1	0	7-bit constant						

SYNTAX: adlk cnst7

EXECUTION: (pc) + 1 $\rightarrow$ pc
(acc)+(7-bit constant) $\rightarrow$ (acc)

WORDS: 1

CYCLES: 1

adll **add to low acc. Immediate.**

BIT:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	0	0	0	1	0							
	16-bit constant															

SYNTAX: adll cnst16
EXECUTION: (pc) + 2 $\rightarrow$ pc
(acc)+(16-bit constant with optional MSBs sign extension*) $\rightarrow$ (acc)

WORDS: 2

CYCLES: 2

Note:option is controlled by CTRL :SENSE bit

and **and with high acc.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	see note 1						

SYNTAX: and dma7
and *(,nar)

EXECUTION: (pc) + 1→ pc
(acc(31:16)) .and. (dma) → (acc(31:16))

WORDS: 1

CYCLES: 1(DI) 2(DE)

andk **and short immediate with high acc.**

BIT:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	7-bit constant						

SYNTAX: andk cnst7

EXECUTION: (pc) + 1 → pc
(acc(23:16) .and. (7-bit constant) → (acc(23:16))
0 → acc(31:24)

WORDS: 1

CYCLES: 1

andl **and immediate with high acc.**

BIT:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	0	1	0	1	0							

16-bit constnat

SYNTAX: andl cnst16

EXECUTION:
(pc) + 2 $\rightarrow$ pc
(acc(31:16)) .and. (16-bit constant) $\rightarrow$ (acc(31:16))

WORDS: 2

CYCLES: 2

apac **add p register to acc.**

BIT:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	1	0	0	1	0	0							

SYNTAX: apac

EXECUTION:
(pc) + 1 $\rightarrow$ pc
(acc)+(p) $\rightarrow$ (acc)

WORDS: 1

CYCLES: 1

bacc **branch to address specified by acc.**

BIT:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	0	1	0	0							

SYNTAX: bacc

EXECUTION: (acc (31:16)) $\rightarrow$ pc

WORDS: 1

CYCLES: 2

bit **test bit.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	bbbb				0	data memory address						

indirect

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	bbbb				1	see note 1						

SYNTAX: bit dma7, bbbb
bit *,bbbb((,nar)

EXECUTION: (pc) + 1 → pc
(dma) → ss(tb)

WORDS: 1

CYCLES: 1(DI) 2(DE)

bs **branch immediate if bit set.**

BIT:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	bbb			0							
program memory address															

SYNTAX: bbb, pma16

EXECUTION: if ss(#1bbb)=1
then (pma) → pc
else (pc)+2 → pc

WORDS: 2

CYCLES: 3

bz **branch immediate if bit reset.**

BIT:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	bbb			0							
program memory address															

SYNTAX: bz bbb, pma16

EXECUTION: if ss(#1bbb)=0
then (pma) → pc
else (pc)+2 → pc

WORDS: 2

CYCLES: 3

cala **call subroutine indirect.**

BIT:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	0							

SYNTAX: cala

EXECUTION: (pc)+1 → (sp)
(acc(31:16))→ pc

WORDS: 1

CYCLES: 2

call **subroutine .**

BIT:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	16-bit constant															

SYNTAX: call pma16

EXECUTION: (pc)+1 $\rightarrow$ (sp)
(16-bit constant)$\rightarrow$ pc

WORDS: 2

CYCLES: 3

dint **disable interrupt.**

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

SYNTAX: dint

EXECUTION: (pc) + 1 $\rightarrow$ pc
1 $\rightarrow$ (INTM) status bit

WORDS: 1

CYCLES: 1

eint **enable interrupt.**

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

SYNTAX: eint

EXECUTION: (pc) + 1 → pc
0 → (INTM) status bit

WORDS: 1

CYCLES: 1

in **input data from port.**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	0	port address			0	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	0	port address			0	1	see note 1						

SYNTAX: in dma7,port
in *,port(,nar)

EXECUTION: (pc) + 1 → pc
port address → a2-a0
(IOPR(4:3)) → a4-a3
0 → a15-a6
(IOR) → dma

WORDS: 1

CYCLES: 1; note:only for internal memory

lac **load acc.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	1	see note 1						

SYNTAX: lac dma7
lac *(,nar)

EXECUTION: (pc) + 1 → pc
(dma) → acc(31:16)
0 → acc(15:0)

WORDS: 1

CYCLES: 1(DI) 2(DE)

lack **load acc. short immediate.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	7-bit sonstant						

SYNTAX: lack cnst7

EXECUTION: (pc) + 1 → pc
(7-bit constant) → acc(23:16)
0 → acc(31:24)
0 → acc(15:0)

WORDS: 1

CYCLES: 1

lacl **load acc. Immediate**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	1	1	0							
16-bit constant															

SYNTAX: lacl cnst16

EXECUTION: (pc) + 2 → pc
(16-bit constant)→ acc(31:16)
0 → acc(15:0)

WORDS: 2

CYCLES: 2

lar **load auxiliary register.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	arp			0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	arp			0	1	see note 1						

SYNTAX: lar dma7, arp
lar *,arp(,nar)

EXECUTION: (pc) + 1 → pc
(dma)→ (ar)

WORDS: 1

CYCLES: 1(DI) 2(DE) ; no manipulation on ars

lark **load auxiliary register short immediate.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	arp			1	0	7-bit constant						

SYNTAX: lark cnst7, arp

EXECUTION: (pc) + 1 → pc
(7-bit constant) → (ar(6:0))
0 → ar(15:7)

WORDS: 1

CYCLES: 1

larl **load auxiliary register immediate.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	0	arp			0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-bit constant															

SYNTAX: larl cnst16, arp

EXECUTION: (pc) + 2 → pc
(16-bit constant)→ ar (15:0)

WORDS: 2

CYCLES: 2

ldp **load data-page register.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	1	see note 1						

SYNTAX: ldp dma7
ldp *(,nar)

EXECUTION: (pc) + 1 → pc
(dma(3:0))→ (dp(3:0))

WORDS: 1

CYCLES: 1(DI) 2(DE)

ldpk **load short immediate to datapage register.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	0	x	x	x	4-bit constant			

SYNTAX: ldpk cnst4

EXECUTION: (pc) + 1 → pc
(4-bit constant)→ (dp(3:0))

WORDS: 1

CYCLES: 1

lip **load io page register**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	1	see note 1						

SYNTAX: lip dma7
lip *(,nar)

EXECUTION: (pc) + 1 →pc
(dma)→ (iop(1:0))

WORDS: 1

CYCLES: 1(DI) 2(DE)

lipk **load iopage register with short immediate.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1	0	x	x	s1	s0	x	x	x

SYNTAX: lipk cnst2

EXECUTION: (pc) + 1→ pc
s1→ iop(1), s0 → iop(0)

WORDS: 1

CYCLES: 1

lup **loop instruction.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	loop number			0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	loop number			0	1	see note 1						

SYNTAX: lup dma, lic
lup *,lic(,nar)

EXECUTION: (pc) + 1 $\rightarrow$ pc
(dma)$\rightarrow$ (rc)
(loop number) $\rightarrow$ (loop counter)

WORDS: 1

CYCLES: 1(DI) 2(DE); the next (loop number+1) words will be repeat (rc+1) times

lupk **load rc with 7-bit constant and enable loop operation.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	loop number			1	0	7-bit constant						

SYNTAX: lupk cnst7, lic

EXECUTION: (pc) + 1 $\rightarrow$ pc
(7-bit constant) $\rightarrow$ (rc)
(loop number) $\rightarrow$ (loop counter)

WORDS: 1

CYCLES: 1

lx **load x register.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0	1	see note 1						

SYNTAX: lx dma7
lx *(,nar)

EXECUTION: (pc) +1 → pc
(dma) → (xr)

WORDS: 1

CYCLES: 1(DI) 2(DE)

lxk **load short immediate to x register.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	0	7-bit constant						

SYNTAX: lxk cnst7

EXECUTION: (pc) + 1 → pc
(7-bit constant) → xr(6:0)
0 → xr(15:0)

WORDS: 1

CYCLES: 1

lxl **load immediate to x register.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0							
16-bit constant															

SYNTAX: lxl cnst16

EXECUTION: (pc) + 2 → pc
(16-bit constant)→ (xr)

WORDS: 2

CYCLES: 2

lxa **load x register and accumulate previous product.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	0	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	0	0	1	see note 1						

SYNTAX: lxa dma7
lxa *(,nar)

EXECUTION: (pc) + 1 → pc
(dma)→ (xr)
(acc)+(P) → (acc)

WORDS: 1

CYCLES: 1(DI) 2(DE)

lxak **load short immediate to x register and accumulate previous product.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	1	0	1	0	7-bit constant						

SYNTAX: lxak cnst7

EXECUTION:
(pc) + 1 → pc
(7-bit constant) → xr(6:0)
0 → xr(15:8)
(acc)+(p) → acc

WORDS: 1

CYCLES: 1

lxal **load immediate to x register and accumulate previous product.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	1	0	1	1	0							
	16-bit constant															

SYNTAX: lxal cnst16

EXECUTION:
(pc) + 2 → pc
(16-bit constant)→ (xr)
(acc)+(p) → acc

WORDS: 2

CYCLES: 2

lxs **load x register and subtract previous product.**

direct:

15	14	13	12	11	10	9	8	7	6 5 4 3 2 1 0
0	0	1	1	0	1	0	0	0	data memory address

indirect:

15	14	13	12	11	10	9	8	7	6 5 4 3 2 1 0
0	0	1	1	0	1	0	0	1	see note 1

SYNTAX: lxs dma7
lxs *(,nar)

EXECUTION: (pc) + 1 → pc
(dma)→ xr
(acc)-(p)→ (acc)

WORDS: 1

CYCLES: 1(DI) 2(DE)

lxsk **load short immediate to x register and subtract previous product.**

Bit:

15	14	13	12	11	10	9	8	7	6 5 4 3 2 1 0
0	0	1	1	0	1	0	1	0	7-bit constant

SYNTAX: lxsk cnst7

EXECUTION: (pc) + 1 → pc
(7-bit constant) → xr(6:0)
0 → xr(15:8)
(acc)- (p) → acc

WORDS: 1

CYCLES: 1

lxsl **load immediate to x register and subtract previous product.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	1	0							
16-bit constant															

SYNTAX: lxsl cnst16

EXECUTION:
(pc) + 2 → pc
(16-bit constant)→ (xr)
(acc)- (p) → acc

WORDS: 2

CYCLES: 2

lxm **load x register and store p register to acc.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0	1	see note 1						

SYNTAX:
lxm dma7
lxm *(,nar)

EXECUTION:
(pc) + 1 → pc
(dma)→ (xr)
(p) → (acc)

WORDS: 1

CYCLES: 1(DI) 2(DE)

lxmk **load short immediate to x register and store p register to acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	1	1	1	0	7-bit constant						

SYNTAX: lxmk cnst7

EXECUTION:
(pc) + 1 → pc
(7-bit constant) →xr(6:0)
0 → xr(15:8)
(p) → acc

WORDS: 1

CYCLES: 1

lxml **load immediate to x register and store p register to acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	1	1	0	0	0							
	16-bit constant															

SYNTAX: lxml cnst16

EXECUTION:
(pc) + 2 → pc
(16-bit constant)→ (xr)
(p) → acc

WORDS: 2

CYCLES: 2

mba **multiply and accumulate previous product.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	0	0	0	0	0	r	i	0	note 2		

SYNTAX: mba riB,*

EXECUTION:
(pc) + 1 → pc
(acc)+(p) → acc
(Mb0(addressed by arp(7:1)*(r)))*(mb1(addressed by arb(7:1)*(i)))→p

WORDS: 1

CYCLES: 1; note: r=0/1, i=0/1

mbs **multiply and subtract previous product.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	0	0	0	0	0	0	r	i	0	note 2		

SYNTAX: mbs ri,*

EXECUTION:
(pc) + 1 $\rightarrow$ pc
(acc)-(p)$\rightarrow$ acc
(mb0(addressed by arb(7:1). (r)))* (mb1(addressed by arb(7:1). (i))) $\rightarrow$ (p)

WORDS: 1

CYCLES: 1; note : r=0/1, i=0/1

mar **modify auxiliary register.**

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	0	1	1	0	1	see note 1						

SYNTAX: MAR *(,nar)

EXECUTION:
(pc) + 1 $\rightarrow$ pc
modifies arp, ar(arp) as specified by the indirect addressing field

WORDS: 1

CYCLES: 1

mod **load modulo register.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	0	0	data memory address						

indirect

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	0	1	see note 1						

SYNTAX: mod dma7
mod *(,nar)

EXECUTION: (pc) + 1 → pc
(dma(6:0))→ mr(6:0)

WORDS: 1

CYCLES: 1(DI) 2(DE)

modk **load modulo register short immediate.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	0	7-bit constant						

SYNTAX: modk cnst7

EXECUTION: (pc) + 1 → pc
(7-bit constant)→ mr(6:0)

WORDS: 1

CYCLES: 1

mpa **array multiplication.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	1	see note 1						

SYNTAX: MPA *(,nar)

EXECUTION:

(pc) + 1 → sp
(acc)→ (pc)
0 → acc
do
begin
    (mb0(addressed by arb(7:0))) * (mb1(addressed by pc(7:0))) → (p)
    (acc)+ (p) → acc
    (rc) - 1 → rc
    (pc) + 1 → pc
end ((rc) + 1) times
(sp)→ pc

WORDS: 1

CYCLES: 3+ (rc); enable repeat operation.

mb **ram_bank_0 multiply ram_bank_1.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	0	0	0	r	i	0	note 2		

SYNTAX: mpc ri,*

EXECUTION:

(pc) + 1 → pc
(mb0(addressed by arb(7:1) . (r))) * (mb1(addressed by arb(7:1) . (i))) → (p)

WORDS: 1

CYCLES: 1; note: r=0/1, i=0/1

mx **(x) multiply (dma)**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	0	1	see note 1						

SYNTAX: mx dma7
mx *(,nar)

EXECUTION: (pc) + 1 $\rightarrow$ pc
(x) * (dma) $\rightarrow$ (p)

WORDS: 1

CYCLES: 1(DI) 2(DE)

mxk **(x) multiply (7-bit constant).**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	1	0	7-bit constnat						

SYNTAX: mxk cnst7

EXECUTION: (pc) + 1 $\rightarrow$ pc
(x)* (7-bit constant) $\rightarrow$ (p)

WORDS: 1

CYCLES: 1

mxl **(x) multiply (16-bit constant).**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	0	0							
16-bit constant															

SYNTAX: mxl cnst16

EXECUTION:
(pc) + 1→ pc
(x) * (16-bit constant) → (p)

WORDS: 2

CYCLES: 2

mxa **(x) multiply (dma) and accumulate previous product.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	1	see note 1						

SYNTAX:
mxa dma7
mxa *(,nar)

EXECUTION:
(pc) + 1 → pc
(acc) + (p) → acc
(x) * (dma) → (p)

WORDS: 1

CYCLES: 1(DI) 2(DE)

mxak **(x) multiply (7-bit constant) and accumulate previous product.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	1	0	1	1	0	7-bit constant						

SYNTAX: mxak cnst7

EXECUTION:
(pc) + 1 $\rightarrow$ pc
(acc) + (p) $\rightarrow$ acc
(x) * (7-bit constant) $\rightarrow$ (p)

WORDS: 1

CYCLES: 1

mxs **(x) multiply (dma) and subtract previous product.**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	0	0	1	0	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	0	0	1	0	1	see note 1						

SYNTAX:
mxs dma7
mxs *(,nar)

EXECUTION:
(pc) + 1 $\rightarrow$ pc
(acc) - (p) $\rightarrow$ acc
(x) * (dma) $\rightarrow$ (p)

WORDS: 1

CYCLES: 1(DI) 2(DE)

mxsk **(x) multiply (7-bit constant) and subtract previous product.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1	0	0	1	1	0	7-bit constant						

SYNTAX: mxsk cnst7

EXECUTION:
(acc) - (p) → acc
(x) * (7-bit constant) → (p)

WORDS: 1

CYCLES: 1

nop **no operation.**

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

SYNTAX: nop

EXECUTION: (pc) + 1 → pc

WORDS: 1

CYCLES: 1

or **or with high acc.**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	1	see note 1						

SYNTAX:
or dma7
or *(,nar)

EXECUTION:
(pc) + 1 → pc
(acc(31:16)).or. (dma) → (acc(31:16))

WORDS: 1

CYCLES: 1(DI) 2(DE)

ork **or short immediate with high acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	1	0	7-bit constant						

SYNTAX: ork cnst7

EXECUTION:
(pc) + 1 → pc
(acc(23:16)).or. (7-bit constant)→ (acc(23:16)
(acc(31:24)) → acc(31:24)

WORDS: 1

CYCLES: 1

orl **or immediate with high acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	0	1	0	0	0							
	16-bit constant															

SYNTAX: orl cnst16

EXECUTION:
(pc) + 2 → pc
(acc(31:16)) .or. (16-bit constant)→ (acc(31:16)

WORDS: 2

CYCLES: 2

out **output data to port.**

direct:

15	14	13	12	11 10 9	8	7	6 5 4 3 2 1 0
0	1	0	0	port address	0	0	data memory address

indirect:

15	14	13	12	11 10 9	8	7	6 5 4 3 2 1 0
0	1	0	0	port address	0	1	see note 1

SYNTAX:
out dma7, port
out port *(,nar)

EXECUTION:
(pc) + 1 → pc
(pa) → address bus a1-a0
(IOPR)(4:3) → a4-a0
0 → a15-a5

WORDS: 1

CYCLES: 1;note: For internal memory only

outk **output short immediate to port.**

Bit:

15	14	13	12	11 10 9	8	7	6 5 4 3 2 1 0
0	1	0	0	port address	1	0	7-bit constant

SYNTAX:
outk cnst7, port

EXECUTION:
(pc) + 1 → pc
(pa) → address bus a2-a0
(IOPR)(4:3) → a4-a3
0 → a15-a5
(7-bit constant)→ IOR (addressed by a4-a0)

WORDS: 1

CYCLES: 1; note: For internal memory only

outl **output immediate to port.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	0	port address			0	0	0	0

16-bit constant

SYNTAX: outl cnst16, port

EXECUTION:
(pc) + 1 → pc
(pa) → address bus a2-a0
(IOPR)(4:3) → a4-a3
0 → a15-a5
(16-bit constant) → IOR(addressed by a4-a0)

WORDS: 1

CYCLES: 1;note: for internal memory only

pac **load acc. With p register**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0	0							

SYNTAX: pac

EXECUTION:
(pc) + 1 → pc
(p) → (acc)

WORDS: 1

CYCLES: 1

poph **pop top of stack to high accumulator.**

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

SYNTAX: poph

EXECUTION: (pc) + 1 → pc
(tos) → acc(31:16)

WORDS: 1

CYCLES: 1

popl **pop top of stack to low accumulator.**

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

SYNTAX: popl

EXECUTION: (pc) + 1 → pc
(tos) → acc(15:0)

WORDS: 1

CYCLES: 1

pop **pop top of stack to data memory.**

	15	14	13	12	11	10	9	8	7	6–0
direct:	1	0	1	1	0	1	0	1	0	data memory address
indirect:	1	0	1	1	0	1	0	1	1	see note 1

SYNTAX: pop dma
pop *(,nar)

EXECUTION: (pc) + 1 → pc
(tos) → dma

WORDS: 1

CYCLES: 1(DI) 2(DE)

psh **push data memory value onto stack.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	0	1	see note 1						

SYNTAX: psh dma
psh *(,nar)

EXECUTION: (pc) + 1 $\rightarrow$ pc
dma $\rightarrow$ (tos)

WORDS: 1

CYCLES: 1

pshh **push high accumulator onto stack.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	0	1	see note 1						

SYNTAX: pshh

EXECUTION: (pc) + 1 $\rightarrow$ pc
acc(31:16) $\rightarrow$ (tos)

WORDS: 1

CYCLES: 1

pshl **push low accumulator onto stack.**

Bits:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	1	1	see note 1						

SYNTAX: pshl

EXECUTION: (pc) + 1 $\rightarrow$ pc
acc(15:0) $\rightarrow$ (tos)

WORDS: 1

CYCLES: 1

ret **return from subroutine.**

Bits:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0							

SYNTAX: ret

EXECUTION: (sp) $\rightarrow$ pc
sp-1 $\rightarrow$ sp

WORDS: 1

CYCLES: 2

reti **return from interrupt.**

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

SYNTAX: reti

EXECUTION:
(sp) $\rightarrow$ pc
(sp)-1 $\rightarrow$ sp
(sp) $\rightarrow$ ss
sp-1$\rightarrow$ sp

WORDS: 1

CYCLES: 2

rovm **reset overflow mode.**

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

SYNTAX: rovm

EXECUTION:
(pc) + 1 $\rightarrow$ pc
0$\rightarrow$ (ovm) status bit.

WORDS: 1

CYCLES: 1

rpt **load repeat counter.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	data memory address						

`

indirect:L

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	1	see note1						

SYNTAX: rpt dam7
rpt *(,nar)

EXECUTION: (pc) + 1 → pc
(dma) → (rc)

WORDS: 1

CYCLES: 1(DI) 2(DE)

rptk **load rc with 7-bit constant.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	0	7-bit constant						

SYNTAX: rptk cnst7

EXECUTION: (pc) + 1 → pc
(7-bit constant) → (rc)

WORDS: 1

CYCLES: 1

rxf **reset external flag.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0							

SYNTAX: rxf

EXECUTION: (pc) + 1 → pc
0 → (XF) pin and status bit.

WORDS: 1

CYCLES: 1

sah **store high acc.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	1	see note 1						

SYNTAX: sah dma7
sah *(,nar)

EXECUTION: (pc) + 1 → pc
(acc(31:16)) → (dma)

WORDS: 1

CYCLES: 1(DI) 2(DE)

sal **store low acc.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1	1	see note 1						

SYNTAX: sal dma7
sal *(,nar)

EXECUTION: (pc) + 1 → pc
(acc(15:0)) → (dma)

WORDS: 1

CYCLES: 1(DI) 2(DE)

sar **store auxiliary register.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	ar			0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	ar			1	see note 1						

SYNTAX: sar dma7, arp
sar *, arp (,nar)

EXECUTION: (pc)+1 → pc
(ar) → (dma)

WORDS: 1

CYCLES: 1(DI) 2(DE)

sbh **subtract from high acc.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	see note 1						

SYNTAX: sbh dma7
sbh *(,nar)

EXECUTION: (pc) +1 → pc
(acc(31:16)) - (dma) → (acc(31:16))

WORDS: 1

CYCLES: 1(DI) 2(DE)

sbhk **subtract short immediate from high acc.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	7-bit constant						

SYNTAX: sbhk cnst7

EXECUTION: (pc)+1 → pc
(acc(31:16)) - (7-bit constant) → (acc(31:16))

WORDS: 1

CYCLES: 1

sbhl **subtract immediate from high acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	0	0	1	0	0							
	16-bit constant															

SYNTAX: sbhl cnst16

EXECUTION: (pc)+2 $\rightarrow$ pc
(acc(31:16)) - (16-bit constant) $\rightarrow$ (acc(31:16))

WORDS: 2

CYCLES: 2

sbl **subtract from low acc.**

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
direct:	0	0	0	0	0	1	1	0	0	data memory address						
indirect:	0	0	0	0	0	1	1	0	1	see note 1						

SYNTAX: sbl dma7
sbl *(,nar)

EXECUTION: (pc)+1 $\rightarrow$ pc
(acc) - (dma with optional MSBs sign extension*) $\rightarrow$ (acc)

WORDS: 1

CYCLES: 1(DI) 2(DE) ; note : Option is controlled by CTRL : SENSE bit

sblk **subtract short immediate from low acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	1	0	7-bit constant						

SYNTAX: sblk cnst7

EXECUTION:
(pc)+1 → pc
(acc) - (7-bit constant) → (acc)

WORDS: 1

CYCLES: 1

sbll **subtract immediate from low acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	0	0	1	1	0							
	16-bit constant															

SYNTAX: sbll cnst16

EXECUTION:
(pc)+2 → pc
(acc) - (16-bit constant with optional MSBs sign extension*) → (acc)

WORDS: 2

CYCLES: 2 ; note:Option is controlled by CTRL: SENSE bit

sdp **store data_page register.**

direct:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1	0	0	0	data memory address						

indirect:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1	0	0	1	see note 1						

SYNTAX: sdp dma7
sdp *(,nar)

EXECUTION: (pc)+1 → pc
(dp) → (dma)

WORDS: 1

CYCLES: 1(DI) 2(DE)

sfl **shift acc left.**

Bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	1	0	0	0	0	shift			

SYNTAX: sfl cnst4

EXECUTION: (pc)+1 → pc
if (shift>< 0)
then
acc * (2** shift)→ acc
else
acc*(2**(sv(3:0))) → acc

WORDS: 1

CYCLES: 1 ; note:15-bit overflow protection.

sfr/sfrs **shift acc right.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	1	1	1	1	0	0	0	0	s	shift			

SYNTAX: sfr cnst4
sfrs cnst4

EXECUTION:
(pc)+1 → pc
if (shift>< 0)
then
acc * (2**(-shift))→ acc
else
acc*(2**(-sv(3:0)) → acc
* s=0 the msbs zero-filled
* s=1 the msbs sign-extended

WORDS: 1

CYCLES: 1

sip **store io_page register**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	0	0	1	0	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	0	0	1	0	1	see note 1						

SYNTAX: sip dma7
sip *(,nar)

EXECUTION:
(pc)+1 → pc
IOPR(4:3)→ (dma (1:0))

WORDS: 1

CYCLES: 1(DI) 2(DE)

sovm **set overflow mode.**

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

SYNTAX: sovm

EXECUTION: (pc)+1 $\rightarrow$ pc
1$\rightarrow$ (OVM) status bit.

WORDS: 1

CYCLES: 1

spac **subtract p register from acc.**

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

SYNTAX: spac

EXECUTION: (pc)+1 $\rightarrow$ pc
(acc) - (p) $\rightarrow$ acc

WORDS: 1

CYCLES: 1

sqra **sqra and accumulate previous product.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	0	1	1	1	see note 1						

SYNTAX: sqra dma7
sqra *(,nar)

EXECUTION:
$0 \rightarrow$ acc
do
begin
(dma) * (dma)$\rightarrow$ (p)
(acc)+(p) $\rightarrow$acc
(rc) - 1 $\rightarrow$ rc
(pc) +1 $\rightarrow$ pc
end ((rc) + 1) times

WORDS: 1

CYCLES: 3+ (rc) ; enable repeat operation.

sss **store ss register.**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	0	0	1	1	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	0	0	1	1	1	see note 1						

SYNTAX: sss dma7
sss *(,nar)

EXECUTION:
(pc)+1 $\rightarrow$ pc
(ss) $\rightarrow$ (dma)

WORDS: 1

CYCLES: 1(DI) 2(DE)

sxf **set external flag.**

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

SYNTAX: sxf

EXECUTION:
(pc)+1 → pc
1 → (XF) pin and status bit.

WORDS: 1

CYCLES: 1

tbr **table read.**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	1	1	0	0	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	1	1	0	0	1	see note 1						

SYNTAX: tbr *(,nar)

EXECUTION:
(pc)+1 → sp
(acc) → pc
do ((pma, addressed by pc) →dma)) ((rc) +1)) times
(sp)→pc

WORDS: 1

CYCLES: 3+(rc) ; enable repeat operation.

trap **software interrupt.**

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

SYNTAX: trap

EXECUTION: (pc)+1 → sp
0c → pc

WORDS: 1

CYCLES: 2

xor **xor with high acc.**

direct:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	0	0	0	data memory address						

indirect:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	0	0	1	see note 1						

SYNTAX: xor dma7
xor *(,nar)

EXECUTION: (pc)+1 → pc
(acc(31:16)) .xor. (dma) → (acc(31:16)

WORDS: 1

CYCLES: 1(DI) 2(DE)

xork **xor short immediate with high acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	0	1	0	7-bit constant						

SYNTAX: xork cnst7

EXECUTION:
(pc)+1 → pc
(acc(23:16)) .xor. (7-bit constant) → (acc(23:16))
(acc(31:24)) → acc(31:24)

WORDS: 1

CYCLES: 1

xorl **xor short immediate with high acc.**

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	0	1	1	0	0							
	16-bit constant															

SYNTAX: xorl cnst16

EXECUTION:
(pc)+2 → pc
(acc(31:16)) .xor. (16-bit constant) → (acc(31:16))

WORDS: 2

CYCLES: 2

* note 1 :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode								1	v	v	v	v		y	

operation: case(vvvv)
0000: no manipulation of ars/arp
0001: y $\rightarrow$ arp
0010: ar(arp) - ar0 $\rightarrow$ ar(arp)
0011: ar(arp) - ar0 $\rightarrow$ ar(arp), y $\rightarrow$ arp
0100: ar(arp) + ar0 $\rightarrow$ ar(arp)
0101: ar(arp) + ar0 $\rightarrow$ ar(arp), y $\rightarrow$ arp
1000: ar(arp) +1 $\rightarrow$ ar(arp)
1001: ar(arp) +1 $\rightarrow$ ar(arp), y $\rightarrow$ arp
1010: ar(arp) - 1 $\rightarrow$ ar(arp)
1011: ar(arp) -1 $\rightarrow$ ar(arp), y $\rightarrow$ arp
1100: ar(arp) +2 $\rightarrow$ ar(arp)
1101: ar(arp) +2 $\rightarrow$ ar(arp), y $\rightarrow$ arp
1110: ar(arp) -2 $\rightarrow$ ar(arp)
1111: ar(arp) -2 $\rightarrow$ ar(arp), y $\rightarrow$ arp

operand:
+ 0
+ 0 ,y
- ARO
- ARO , y
+ ARO
+ ARO , y
+
+ , y
-
- , y
++
++ , y
--
-- , y

* note 2 :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode												x	y	y	y

operation: case(yyy)
000: no operation on ar(arp)
001: ar(arp) - ar0 $\rightarrow$ ar(arp)
010: ar(arp) + ar0 $\rightarrow$ ar(arp)
111: reserved
100: ar(arp) +1 $\rightarrow$ ar(arp)
101: ar(arp) -1 $\rightarrow$ ar(arp)
110: ar(arp) +2 $\rightarrow$ ar(arp)
111: ar(arp) -2 $\rightarrow$ ar(arp)

DC CHARACTERISTICS: TA = 0 to 70∞ C, VCC = 5V ± 10%

SYMBOL	PARAMETER		MIN	TYPE	MAX	UNIT
VCC	Supply voltage		4.5	5	5.5	V
VOL	Supply voltage			0		V
VOH	Output high voltage			4		V
VOL	Output low voltage			0.3	0.6	V
VIH	Input high voltage	BIO(7:0), ED(15:0), HOLD\, EROM(schmite-trigger)	3.5			V
		all others		2.0		V
VIL	Input low voltage	BIO(7:0), ED(15:0), HOLD\, EROM(schmite-trigger)	-0.1	1.5		V
		all others			0.8	V
IOLA	Output low current type A		4	OA		mA
IOLB	Output low current type B		4	OB		mA
IOLC	Output low current type C		16	OC		mA
IOHA	Supply high current (HOLD\)		2			mA
IOHB	Supply high current(HOLD\)		2			mA
IOHC	Supply high current(HOLD\)		8			mA
ICC	Supply current(HOLD\)			10		mA

AC CHARACTERISTICS:

ROM/RAM/IO READ/WRITE TIMING

SYMBOL	PARAMETER	MIN	NOM	MAX	UNIT
Tcs	Chip select access time (ROM, RAM, IO)	25+wTc			ns
Taa	Address access time (ROM, RAM, IO)	25+wTc			ns
Trds	Data set-up time before ERD\ high (ROM, RAM, IO)	12			ns
Twds	Data set-up time before EWR\ high (ROM, RAM, IO)	12			ns
Tdh	Data hold time after ERD\/EWR\ high (ROM, RAM, IO)	0			ns
Tah	Address hold time after ERD\/EWR\ high (ROM, RAM, IO)	0			ns
Ts(a-w)	Address set-up time before EWR\		0-5		ns
Ts(a-r)	Address set-up time before ERD\				

0-5 ns

OUTPUT PORTS AND EXTERNAL FLAG (XF\) TIMING

SYMBOL	PARAMETER	MIN	NOM	MAX	UNIT
Td (a-o)	Address to output ports (xf\) delay time	0		10	ns

RESET TIMING

SYMBOL	PARAMETER	MIN	NOM	MAX	UNIT
Tw (rst)	Reset low pulse width	3Tc			

AC CHARACTERISTICS: *(Continued)*

CLOCK TIMING

SYMBOL	PARAMETER	MIN	NOM	MAX	UNIT
Tc(c)	CLKIN cycle time	30		42	ns
Tlpd(c)	CLKIN low pulse duration(tc=30ns)	12		18	ns
Thpd(c)	CLKIN high pulse duration (tc=30ns)	12		18	ns
Td(c-m)	CLKIN to MCO delay time	0		15	ns

CODEC TRANSMIT AND RECEIVE TIMING

SYMBOL	PARAMETER	MIN	NOM	MAX	UNIT
Tc (mck)	MCK cycle time		650		ns
Tlpd (mck)	MCK low pulse duration	315		335	ns
Thpd (mck)	MCK high pulse duration	315		335	ns
Td (ch-fs)	MCK to FS delay time			20	ns
Td (ch-dx)	DX valid after MCK rising edge			10	ns
Ts (dr)	DR set-up time before MCK falling edge	10			ns
Th (dr)	DR hold time before MCK falling edge	10			ns

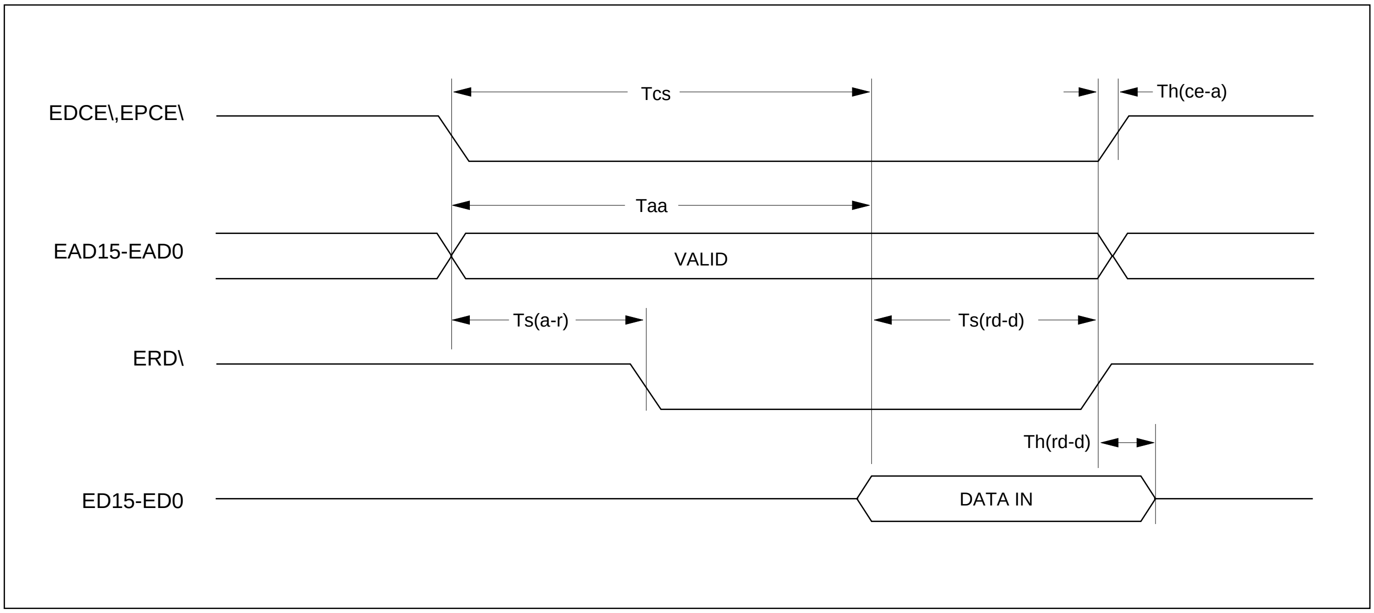
INTERRUPT TIMING

SYMBOL	PARAMETER	MIN	NOM	MAX	UNIT
Tw	INT\ low pulse duration	3Q*		10	ns
Tf	INT\ fall time			10	ns
Ts (int)	INT\ set-up time before MCO falling edge	5		3Q-5	ns

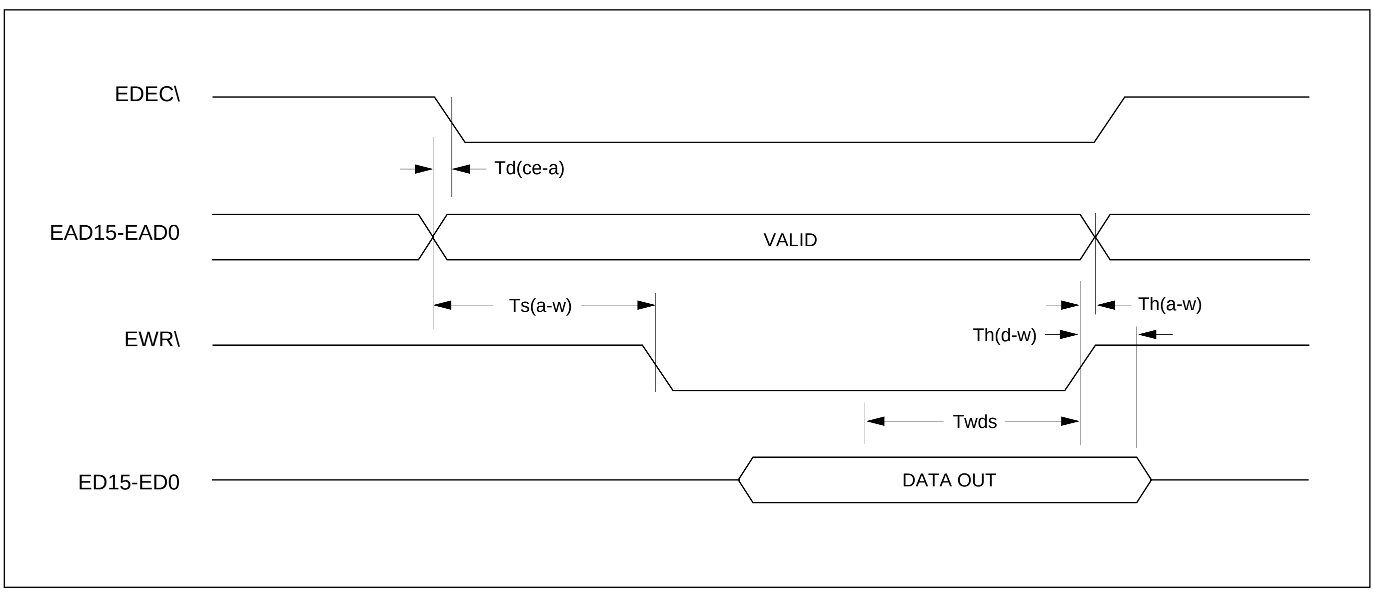
HOLD\TIMING

SYMBO	PARAMETER	MIN	NOM	MAX	UNIT
Ts (a-h)	Address set-up time before HOLD\ low	5		3Q-10	ns
Tdt	Address tri-state after MCO low	1Q-5		1Q+10	ns
Td(al-a)	HOLDA\ low to address tri-state	0			
Td (hh-ha)	HOLD\ high to HOLDA\ high	0	1Q	1Q+10	ns
Ten (ah-a)	Address driven after HOLDA\ high	1Q-10	1Q	2Q	ns

SRAM/ROM READ TIMING



SRAM/WRITE TIMING



AC CHARACTERISTIC: *(Continued)*

DRAM TIMING

SYMBO	PARAMETER	MIN	NOM	MAX	UNIT
Tras	RAS\low pulse duration	10Q-10	10Q		ns
Trp	RAS\ precharge time	7Q-10	7Q		ns
Trcd	RAS\ to CAS\ delay time	4Q-10	4Q		ns
Tcas	CAS\ low pulse duration	6Q-10	6Q		ns
Tcp	CAS\ precharge time	2Q-5	2Q		ns
Tasr	Row address set-up time	1Q-10	1Q		ns
Trah	Row address hold time	3Q-10	3Q		ns
Tasc	Column address hold time	6Q-10	1Q		ns
Qah	Column address hold time	6Q-10	6Q		ns
Td(rd-c)	DRD\ low to CAS\ low	1Q-10	1Q		ns
Td(wr-c)	DWR\ low to CAS\low	1Q-10	1Q		ns
Ts(cas)	Data set-up time before CAS\ high	1Q			ns
Th(cas)	Data hold time after CAS\high	5			ns
Ts(w-ca)	Data set-up time before CAS\low	1Q-10			ns
Th(w-ca)	Data hold time before CAS\low	4Q-10	4Q		ns

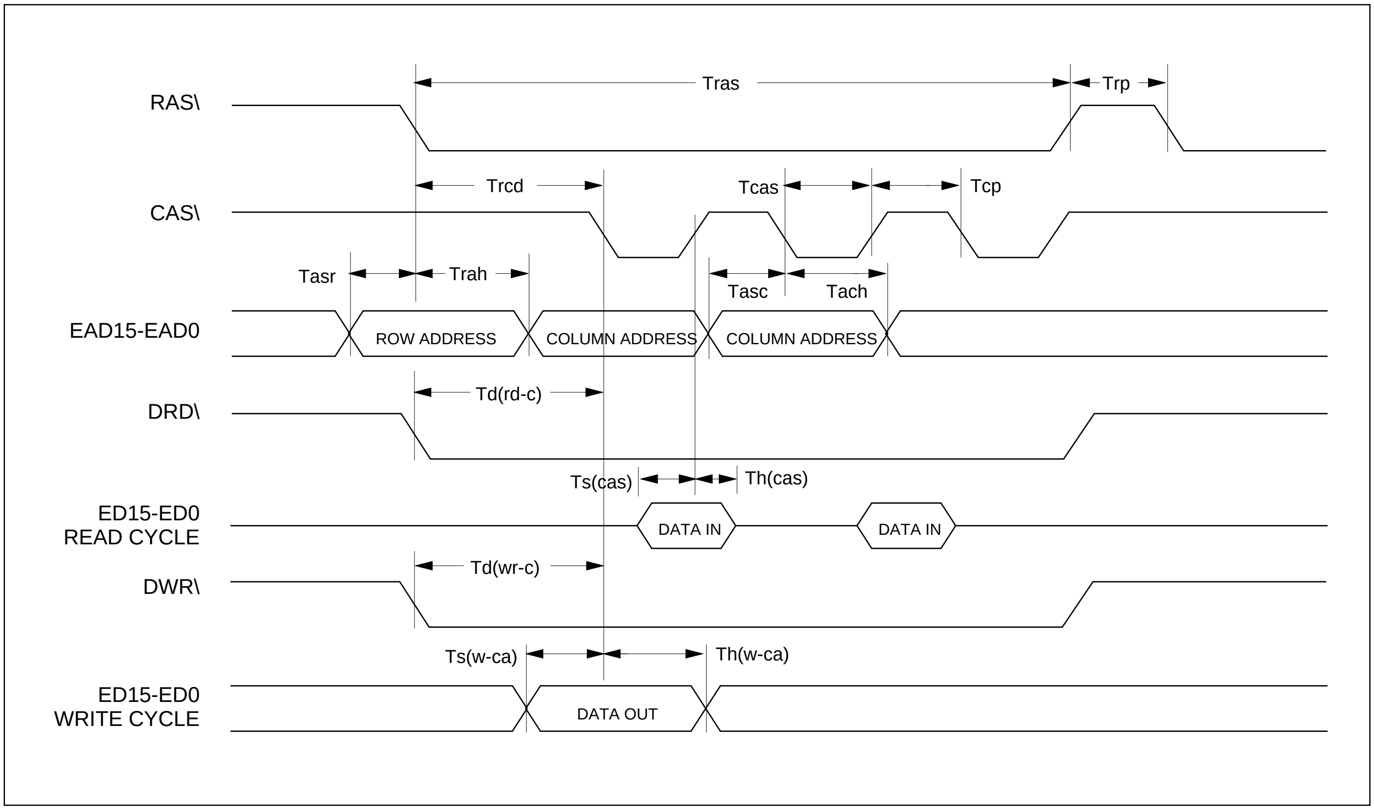
UP INTERFACE

SYMBOL	PARAMETER	MIN	MON	MAX	UNIT
Thra	Host read access time		50		ns
Thdh	Read data hold time	5			ns
Thsw	Write data set up time	20			ns
Thwh	Write data hold time	10			ns

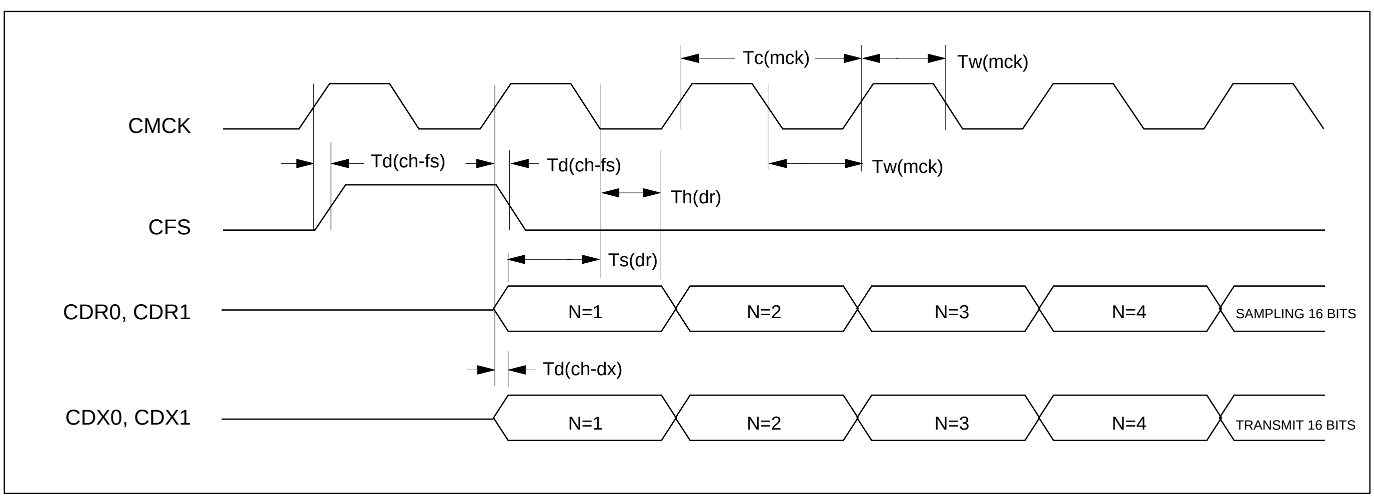
Note:*w=number of wait state

*Q=1/2 TC

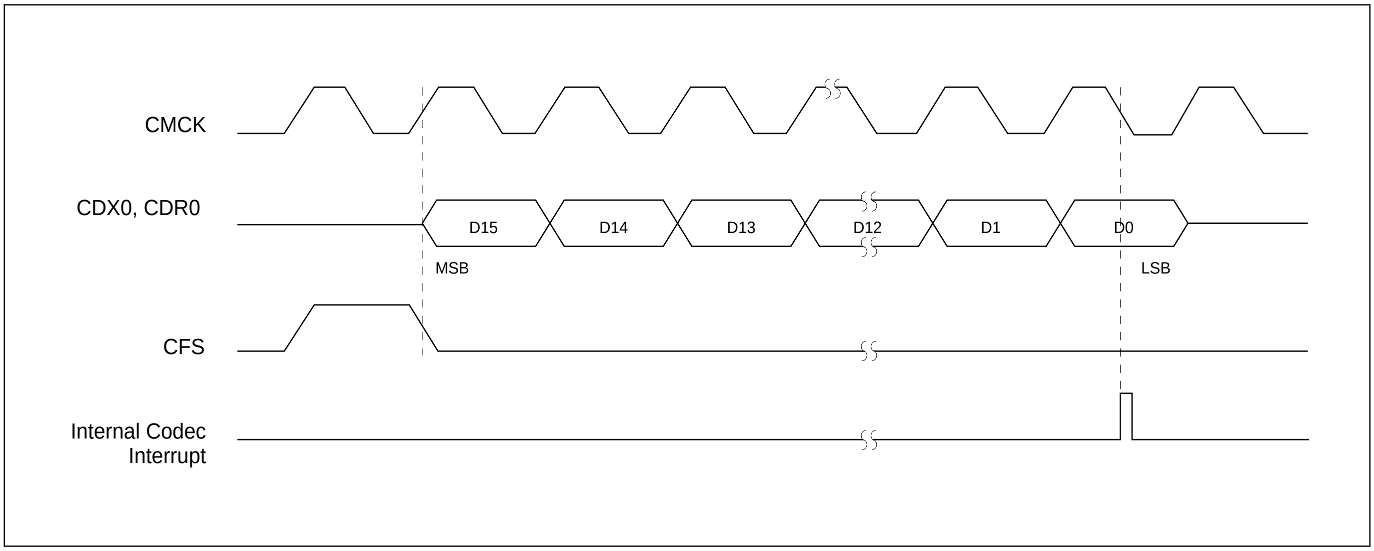
DRAM READ/WRITE TIMING



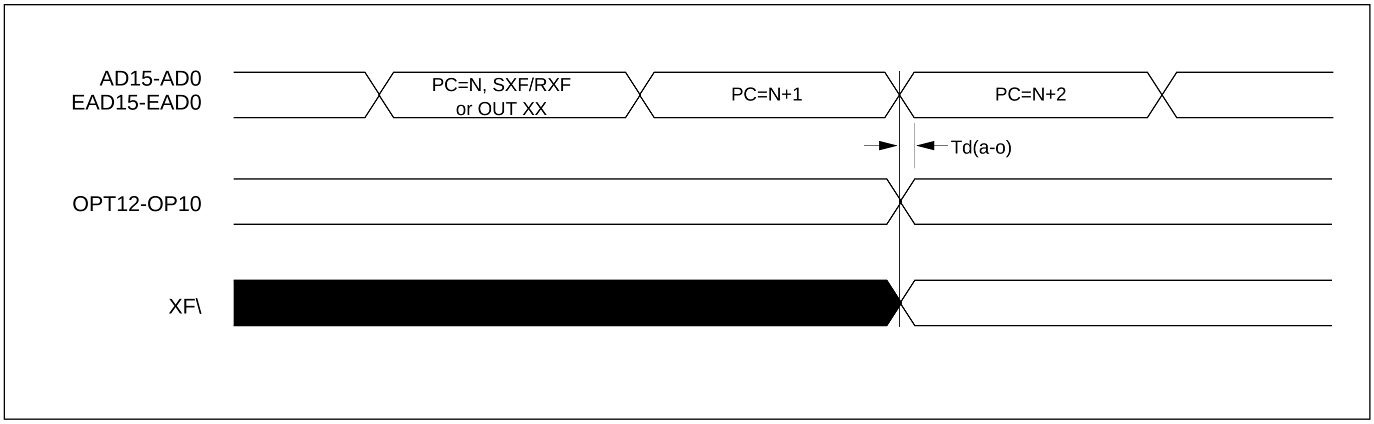
CODEC PORT TRANSMIT AND RECEIVE TIMING



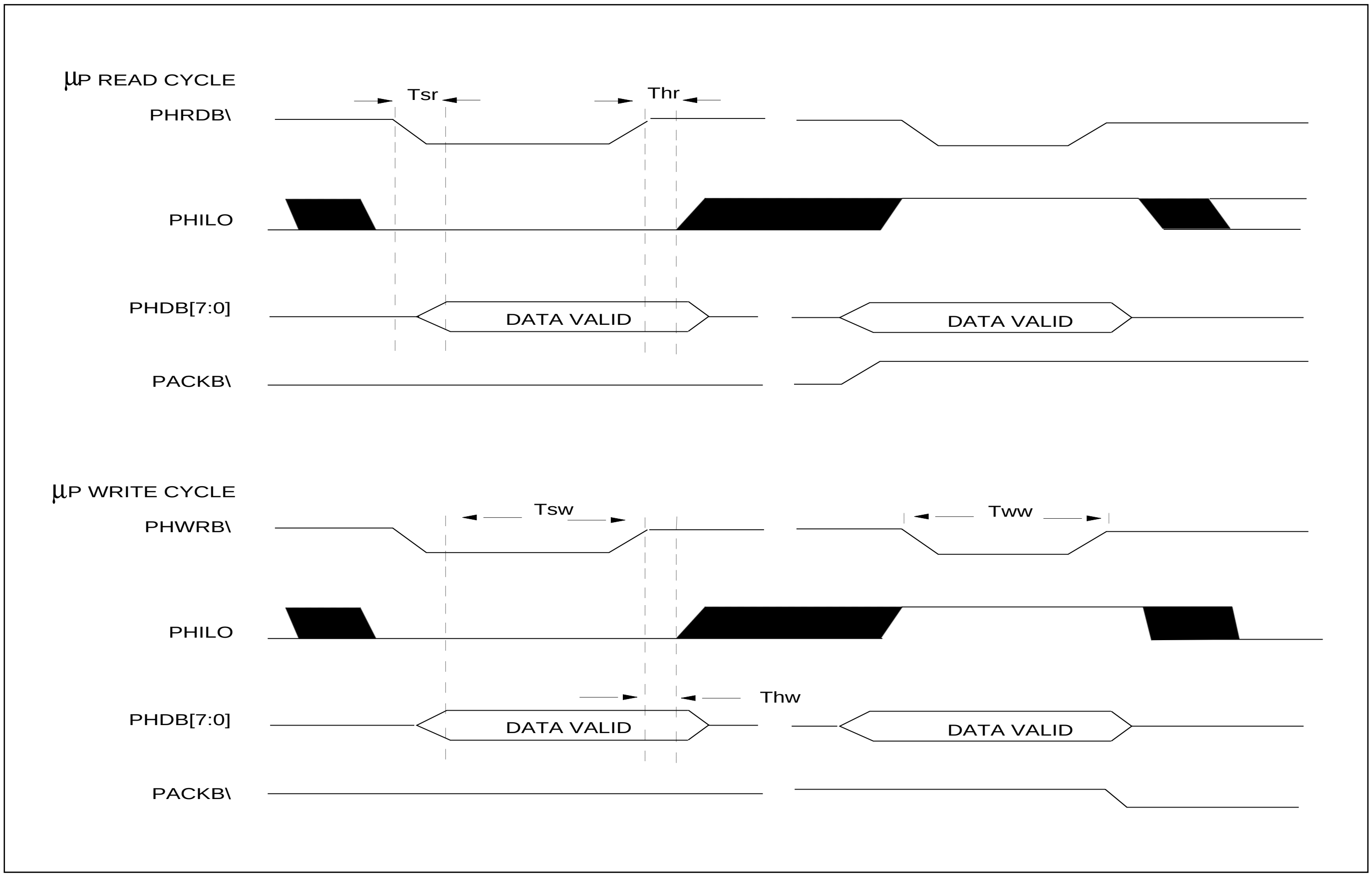
CODEC TRANSMIT RECEIVE OPERATION



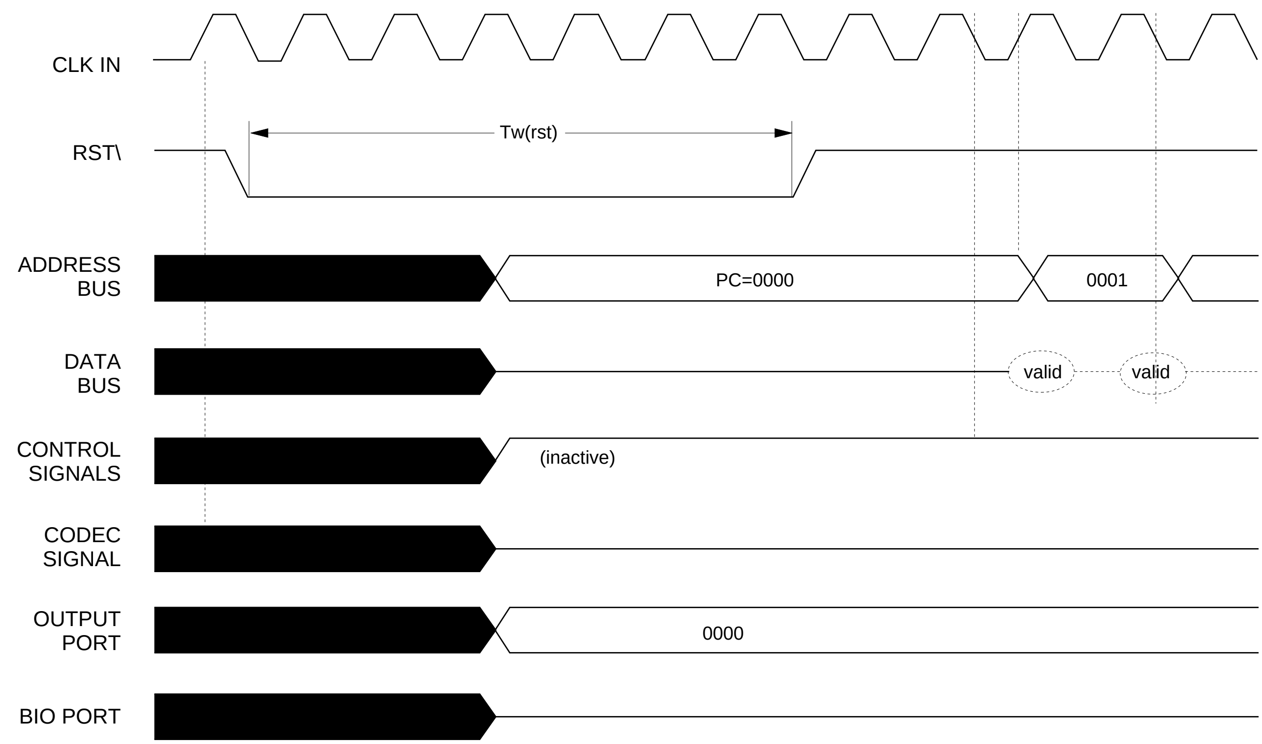
OUTPUT PORT AND EXTERNAL FLAG(XF\) TIMING



µP INTERFACE TIMING

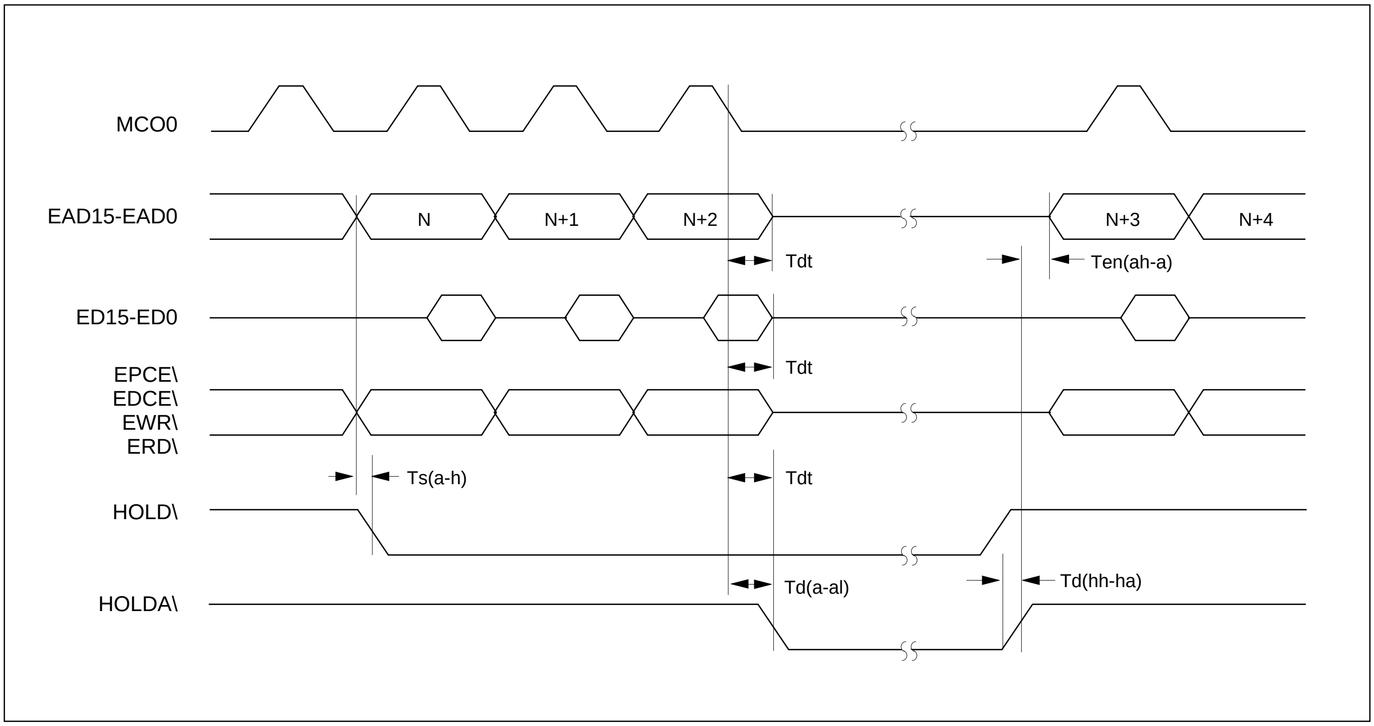


RESET TIMING

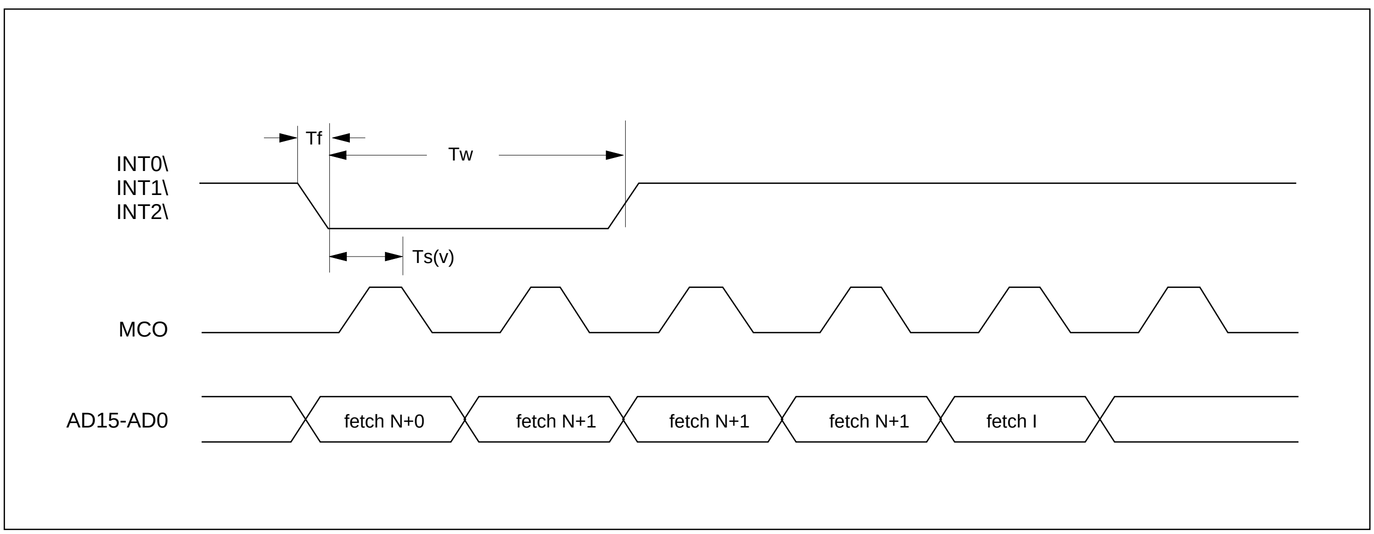


Note: Control Signals
XF\ HOLDA\ EDCE\EPCE\ ERD\ ERD\ EWR\
CAS\ RAS\ DRD\ DWR\

HOLD TIMING

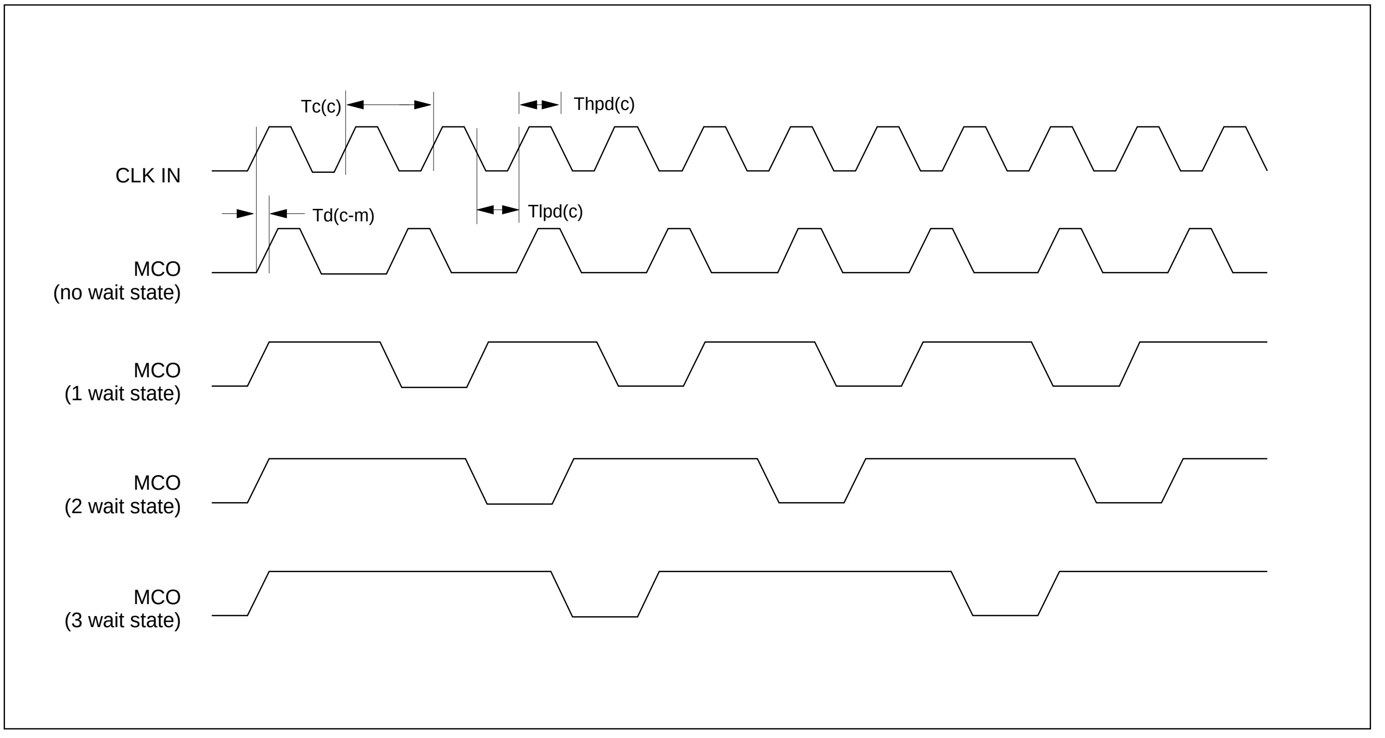


INTERRUPT TIMING



TIMING WAVEFORMS

CLOCK TIMING



ORDERING INFORMATION

PART NO.	PACKAGE
MX96037	PQFP

MX **96** **037** **F** **C**

MX: MXIC COMPONY PREFIX

96: FAMILY PREFIX

037: PRODUCT NUMBER

F: PACKAGE TYPE
F : PQFP
Q : PLCC

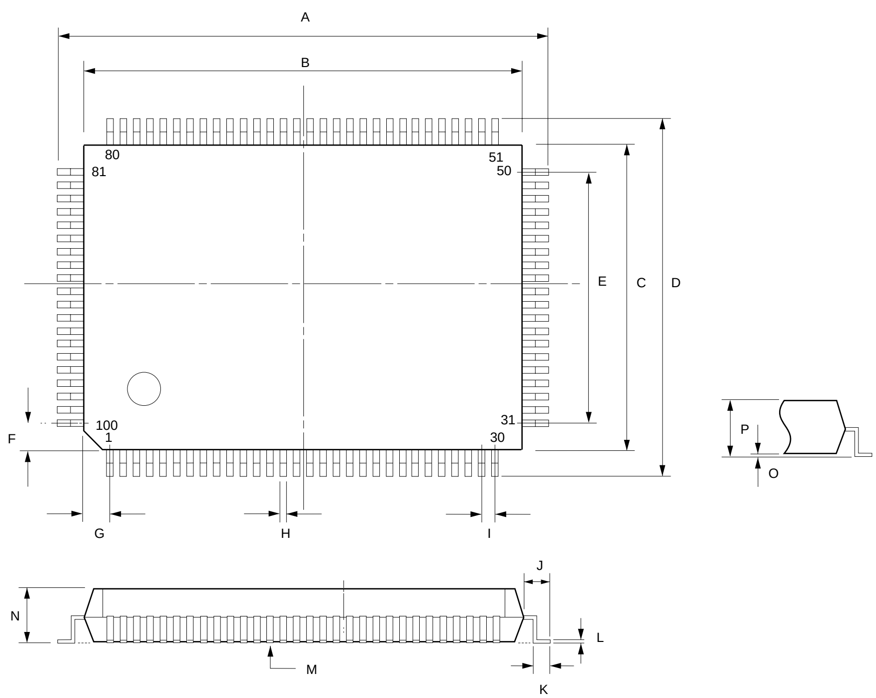
C: COMMERCIAL 0 ~ 70°C

PACKAGE INFORMATION

100-PIN PQFP

ITEM	MILLIMETERS	INCHES
A	24.80 ± .40	.967 ± .016
B	20.00 ± .13	.787 ± .005
C	14.00 ± .13	.551 ± .005
D	18.80 ± .40	.740 ± .016
E	12.35 [REF]	.486 [REF]
F	.83 [REF]	.033 [REF]
G	.58 [REF]	.023 [REF]
H	.30 [Typ.]	.012 [Typ.]
I	.65[Typ.]	.026 [Typ.]
J	2.40 [Typ.]	.094 [Typ.]
K	1.20 [Typ.]	.047 [Typ.]
L	.15 [Typ.]	.006 [Typ.]
M	.10 max.	.004 max.
N	2.75 ± .15	.018 ± .006
O	.10 min.	.004 min.
P	3.30 max.	.103 max.

NOTE: Each lead centerline is located within .25mm[.01 inch] of its true position [TP] at a maximum material condition.



MACRONIX INTERNATIONAL CO., LTD.

HEADQUARTERS:
No. 3, Creation Road III, Science-Based Industrial Park, Hsin Chu, Taiwan, R.O.C.
TEL : +886-3-578-8888
FAX: +886-3-578-8887

TAIPEI OFFICE:
12F, No. 4, Min-Chuan E. Rd., Sec. 3, Taipei, Taiwan, R.O.C.
TEL : +886-2-509-3300
FAX: +886-2-509-2200

EUROPE OFFICE:
Grote Winkellaan 95, Bus 1 1853 Strombeek, Belgium
TEL : +32-2-267-7050
FAX: +32-2-267-9700

SINGAPORE OFFICE:
5 Jalan Masjid Kembangan Court #01-12 Singapore 418924
TEL : +65-747-2309
FAX: +65-748-4090

MACRONIX AMERICA, INC.
1338 Ridder Park Drive, San Jose, CA95131 U.S.A.
TEL : +1-408-453-8088
FAX: +1-408-453-8488

JAPAN OFFICE:
NFK Kawasaki Building, 8F, 1-2 Higashida-cho, Kawasaki-ku
Kawasaki-shi, Kawasaki-ken 210, Japan
TEL : +81-44-246-9100
FAX: +81-44-246-9105