Minnow Processor Manual

Table of Contents

Contents

1 Executive Mode and IO Instructions 4
1.1 Internal IO Instructions 5
1.1.1 APRID – APR Identification 5
1.1.2 WRAPR – Write APR Conditions 7
1.1.3 RDAPR - Read APR conditions 8
1.1.4 SZAPR - Skip on masked APR conditions all zero 9
1.1.5 SNAPR - Skip on any masked APR conditions non-zero 10
1.1.6 WRPI – Write PI Conditions 11
1.1.7 RDPI – Read PI Conditions 12
1.1.8 SZPI - Skip on masked PI conditions all zero 12
1.1.9 SNPI - Skip on any masked PI condition non-zero 13
1.1.10 SETCU - Set CST-update-needed bits 14
1.1.11 RDUBR – Read User Base Register 15
1.1.12 CLRPT – Clear Page Table Entry 16
1.1.13 WRUBR – Write to the User Base Register 17
1.1.14 WREBR – Write to the Exec Base Register 18
1.1.15 RDEBR – Read the Executive Base Register 19
1.1.16 RDSPB - Read SPT Base Register 20
1.1.17 RDCSB – Read CST Base Register 21
1.1.18 RDPUR – Read Process Use Register 22
1.1.19 RDCSTM 23
1.1.20 RDTIM – Read Timebase condition 24
1.1.21 RDINT – Read Interval Timer conditions 25
1.1.22 RDHSB - Read Halt Status Block address 26
1.1.23 WRSPB - Write SPT Base Register 27
1.1.24 WRCSB – Write CST Base Register 28
1.1.25 WRPUR – Write Process Use Register 29
1.1.26 WRCSTM - Write CST Mask Register 30
1.1.27 WRTIM - Write Timebase conditions 31
1.1.28 WRINT - Write Interval Timer conditions 32
1.1.29 WRHSB - Write Halt Status Block Address 33
2 Timers 34
2.1 Time Base 34
2.1.1 Minnow Time Base 34
2.2 Interval Timer Operation 34
2.2.1 Minnow Interval Timer 34
3 ALU 35
4 Pager 36
5 Priority Interrupt Controller 37
6 UART I/O (Local Bus) 39
6.1 UART I/O Memory Map 40
6.2 UART Interrupt Priority Encoder and Status Register 42
6.3 UART Baud Rate Generator 44
6.4 SCN2651 UART Design 44
6.4.1 SCN2651 Register Set 44
6.4.1.1 SCN2651 Receive Holding Register 44
6.4.1.2 SCN2651 Transmit Holding Register 44
6.4.1.3 SCN2651 Status Register (Read Only) 45
6.4.1.4 SCN2651 SYN2/DLE Register (Write Only) 45
6.4.1.5 SCN2651 Mode Register 1 (Read Only) 45
6.4.1.6 SCN2651 Mode Register 2 (Read Only) 45

6.4.1.7 SCN2651 Command Register (Read/Write) 46
7 Microsequencer 47
7.1 Dispatch ROM 48
7.2 Am2910 Microsequencer 48
7.2.1 Am2910 Condition Code Selection. 49
7.2.2 Am2910 Jumps 49
7.2.3 Am2910 Conditional Calls 51
7.2.4 Am2910 Dispatch Operations 51
7.2.5 Am2910 Conditional Returns 51
7.2.6 Am2910 Loops 52
7.3 Control ROM 52
8 RAMFILE 53
8.1 RAMFILE Address Modes 53
8.1.1 Ramfile in Direct Access Mode 54
8.1.2 Ramfile in AC Block | VMA Mode 54
8.1.3 Ramfile in AC Block | AC+N Mode 54
8.1.4 Ramfile in AC Block | XR Mode 54
8.1.5 Ramfile in AC Block | IW Mode 54
8.1.6 Ramfile in Page Table Mode 54

1 Executive Mode and IO Instructions

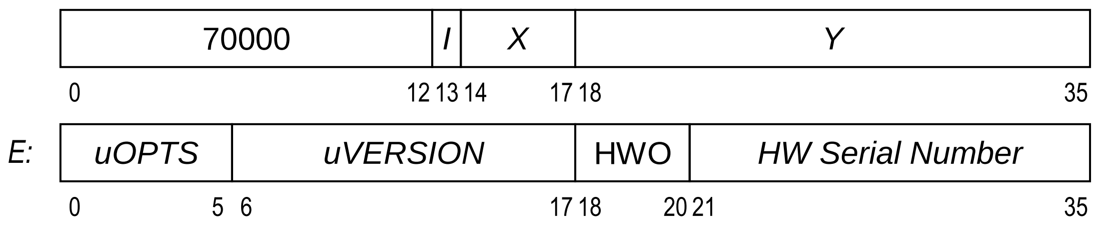
This section of this document describes the operation of the Executive Mode and I/O instructions on the KT-20 (Minnow) platform.

OPCODE Assignment Map								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
700	APR0	APR1	APR2	-	-	-	-	-
710	TIOE	TION	RDI0	WRIO	BSIO	BCIO	-	-
720	TIOEB	TIONB	RDIOB	WRIOB	BSIOB	BCIOB	-	-
730	-	-	-	-	-	-	-	-
740	-	-	-	-	-	-	-	-
750	-	-	-	-	-	-	-	-
760	-	-	-	-	-	-	-	-
770	-	-	-	-	-	-	-	-

AC Field Assignments			
AC	**700**	**701**	**702**
00	APRID	UUO	RDSPB
01	UUO	RDUBR	RDCSB
02	UUO	CLRPT	RDPUR
03	UUO	WRUBR	RDCSTM
04	WRAPR	WREBR	RDTIM
05	RDAPR	RDEBR	RDINT
06	SZAPR	UUO	RDHSB
07	SNAPR	UUO	UUO
10	UUO	UUO	WRSPB
11	UUO	UUO	WRCSB
12	UUO	UUO	WRPUR
13	UUO	UUO	WRCSTM
14	WRPI	UUO	WRTIM
15	RDPI	UUO	WRINT
16	SZPI	UUO	WRHSB
17	SNPI	UUO	UUO

1.1 Internal IO Instructions

1.1.1 APRID – APR Identification



Description:

This instruction returns the microcode version number and the CPU serial number. The word is stored at E as follows:

Bits:

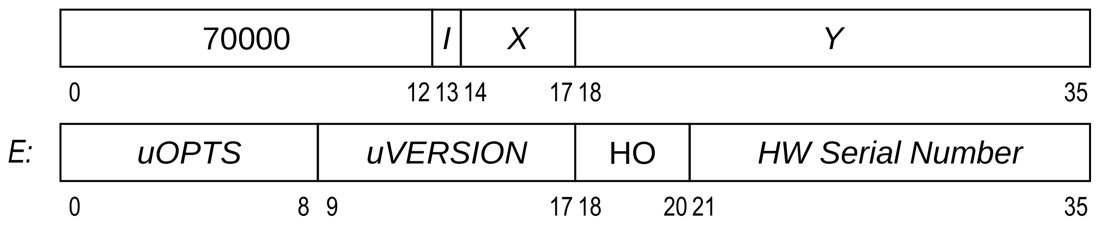
Bit 0 – returned as 0
Bit 1 – 1, if Address Break is compiled into microcode, 0 otherwise
Bit 2 – returned as 0
Bit 3 – returned as 0
Bit 4 – returned as 0
Bit 5 – returned as 0
Bit 6-17, Microcode Version – returned as 4002
Bit 18-20, Hardware Options – returned as 0
Bit 21-35, HW Serial Number – returned as 0

Compatibility:

This is compatible with the KD10 APRID, for what it is worth. It is incompatible with KS10, KL10, KF10, and KC10

Notes:

1. This instruction is incompatible with the KS10. The microcode option field is 3-bits narrower and the microcode version number field is 3 bits wider.
2. The hardware information is returned as zero.
3. The KC10 APRID adds another word of information.



Description:

This instruction returns the microcode version number and the CPU serial number. The word is stored at E as follows:

KS10 Bits:

Bit 0 – INHCST - Inhibit CSD update is available
Bit 1 – NOCST - No CST at all
Bit 2 – NONSTD – Non Standard Microcode
Bit 3 – UBABLT – UPABLT Instructions are available
Bit 4 – KI Paging – KI Paging is present
Bit 5 – KL Paging – KL Paging is present
Bit 6 – Zero
Bit 7 – Zero
Bit 8 – Zero
Bit 9-17, Microcode Version – set to o130
Bit 18-20, Hardware Options – set to 0
Bit 21-35, HW Serial Number – set to o10001 (4097)

KL10 Bits:

Bit 0 – T20. The microcode implements TOPS-20 paging. 0 indicates TOPS-10 paging.
Bit 1 – XA. The microcode implements extended addresses.
Bit 2 – XM. Exotic Microcode.
Bit 3 –
Bit 4 –
Bit 5 –
Bit 6 –
Bit 7 –
Bit 8 –
Bit 9 – 17,
Bit 18 – 50Hz line power.
Bit 19 – Cache is present.
Bit 20 – CHN.
Bit 21 – XAH.
Bit 22 – MOS.
Bit 23 – Keep feature in the paging board
Bit 24–35, HW Serial Number

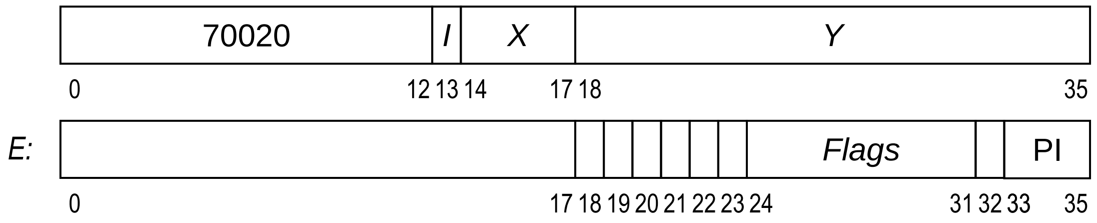
Compatibility:

This is compatible with the KS10 APRID.

Notes:

This is a note

1.1.2 WRAPR – Write APR Conditions



Description:

This immediate instruction decodes its effective address to control the processor. The effective address bits are used as follows:

Bits:

Bit 18 – Unused
Bit 19 – Clear all IO devices (Not present on KS10)
Bit 20 – Enable selected flags
Bit 21 – Disable selected flags
Bit 22 – Clear selected flags
Bit 23 – Set selected flags
Bit 24 – Flag,
Bit 25 – Flag, (Interrupt to console on KS10)
Bit 26 – Flag, (Power Failure on KS10)
Bit 27 – Flag, (No memory (NXM on KS10)
Bit 28 – Flag, (Uncorrectable memory error on KS10)
Bit 29 – Flag, (Correctable memory error on KS10)
Bit 30 – Flag, Interval Timer
Bit 31 – Flag, (Interrupt from console on KS10)
Bit 32 – Should be zero
Bit 33 – 35, PI Level

Compatibility:

This instruction is backward compatible with the KS10. Bit 19 is commonly implemented on Toad, KL10, KT20, KC10 and KF10. Bit 19 is not implemented on KS10, TS10, or SIMH.

Notes:

1. The action of the processor is not defined when both bits 20 and 21 or 22 and 23 are set in the same instruction.
2. As far as I can tell, the only flag that is implemented is the Interval Timer.

1.1.3 RDAPR - Read APR conditions

70024	*I*	*X*	*Y*
0 12	13	14 17	18 35

E:

zero	*Flags*	*zero*	*zero*	*Flags*	I	PIA
0 5	6 13	14 17	18 23	24 31	32	33 35

Description:

This instruction stores the APR status in the word addressed by E. The status is as follows:

Bits:

Bit 06 – Flag Enabled for IRQ,
Bit 07 – Flag Enabled for IRQ,
Bit 08 – Flag Enabled for IRQ,
Bit 09 – Flag Enabled for IRQ,
Bit 10 – Flag Enabled for IRQ,
Bit 11 – Flag Enabled for IRQ,
Bit 12 – Flag Enabled for IRQ, Interval Timer
Bit 13 – Flag Enabled for IRQ,
Bit 24 – Flag,
Bit 25 – Flag,
Bit 26 – Flag,
Bit 27 – Flag,
Bit 28 – Flag,
Bit 29 – Flag,
Bit 30 – Flag, Interval Timer
Bit 31 – Flag,
Bit 32 – Interrupt requested
Bit 33 – 35, PI Level

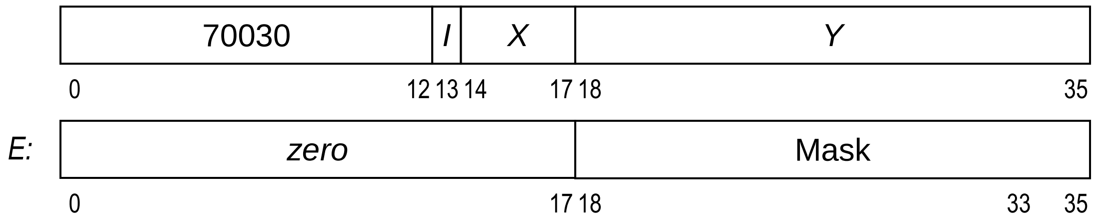
Compatibility:

This instruction is compatible with the KS10.

Notes:

As far as I can tell, the only flag that is implemented is the Interval Timer.

1.1.4 SZAPR - Skip on masked APR conditions all zero



Description:

Test the conditions as returned by RDAPR against the mask produced by bits 18-35 of the effective address. If all masked bits selected by ones in E are zero, skip the next instruction in sequence.

Bits:

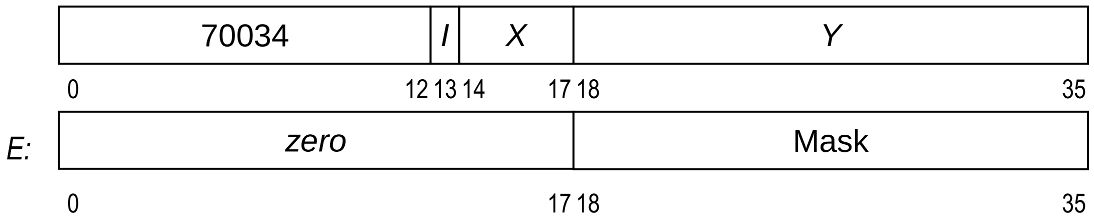
Bits 18-35 - Mask.

Compatibility:

This instruction is compatible with the KS10.

Notes:

1.1.5 SNAPR - Skip on any masked APR conditions non-zero



Description:

Test the conditions as returned by RDAPR against the mask produced by bits 18-35 of the effective address. If any masked bit selected by ones in E is one, skip the next instruction in sequence.

Bits:

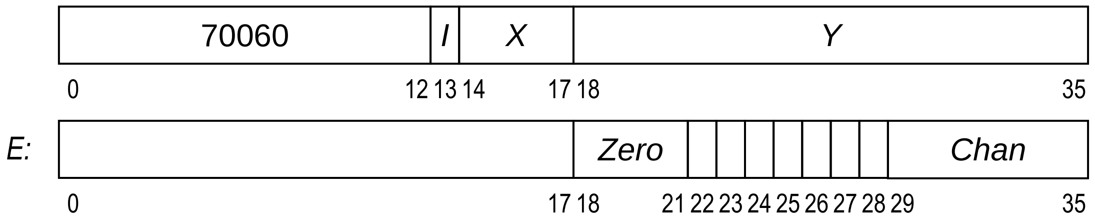
Bits 18-35 - Mask.

Compatibility:

This instruction is compatible with the KS10.

Notes:

1.1.6 WRPI – Write PI Conditions



Description

This instruction configures the Priority Interrupt Controller according to E:

Bits:

Bit 22 – Drop Program Requests on Selected Channels
Bit 23 – Clear PI system
Bit 24 – Initiate Interrupt on Selected Channels
Bit 25 – Turn on the Selected Channels
Bit 26 – Turn off the Selected Channels
Bit 27 – Deactivate the PI system
Bit 28 – Activate the PI system
Bit 29 – 35, Select Channels (See bits 22, 24, 25, and 26)

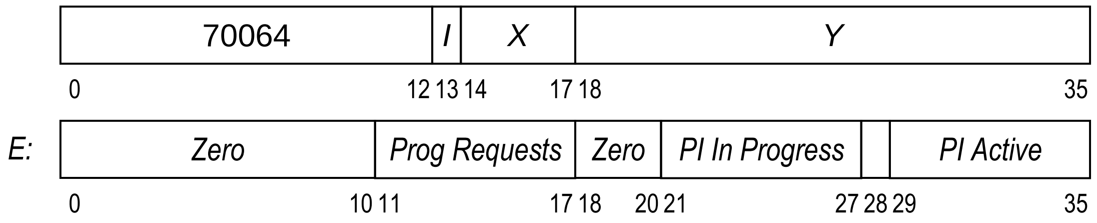
Compatibility*:*

This instruction is identical to the KS10 WRPI Instruction.

Notes:

The action of the processor is not defined when both 25 and 26 or 22 and 24 are set in the same instruction.

1.1.7 RDPI – Read PI Conditions



Description

This instruction stores the PI status in the word addressed by E.

Bits:

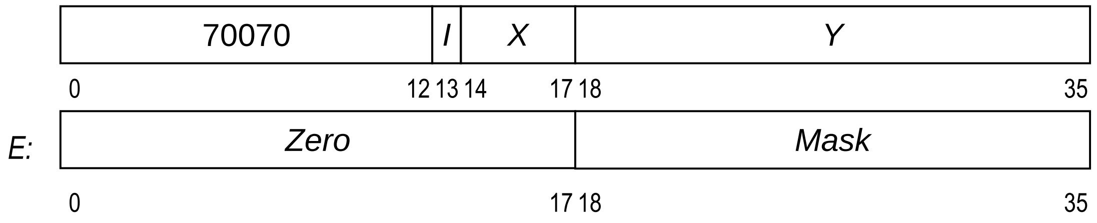
Bit 11-17 – Program Requests
Bit 21-27 – Interrupts in Progress
Bit 28 – PI System On
Bit 29-35 – Active Channels

Compatibility:

This instruction is identical to the KS10 RDPI Instruction.

Notes:

1.1.8 SZPI - Skip on masked PI conditions all zero



Description:

Test the conditions as returned by RDPI against the mask produced by bits 18-35 of the effective address. If any masked bit selected by one in E is one, skip the next instruction in sequence.

Bits:

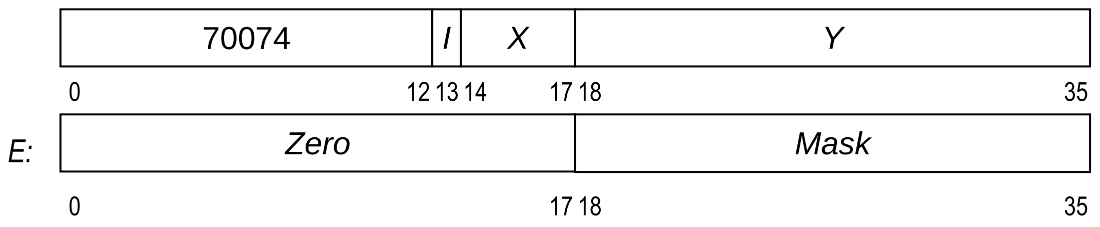
Bits 18-35 - Mask.

Compatibility:

This instruction is compatible with the KS10.

Notes:

1.1.9 SNPI - Skip on any masked PI condition non-zero



Description:

Test the conditions as returned by RDAPR against the mask produced by bits 18-35 of the effective address. If any masked bit selected by ones in E is one, skip the next instruction in sequence.

Bits:

Bits 18-35 - Mask.

Compatibility:

This instruction is compatible with the KS10.

Notes:

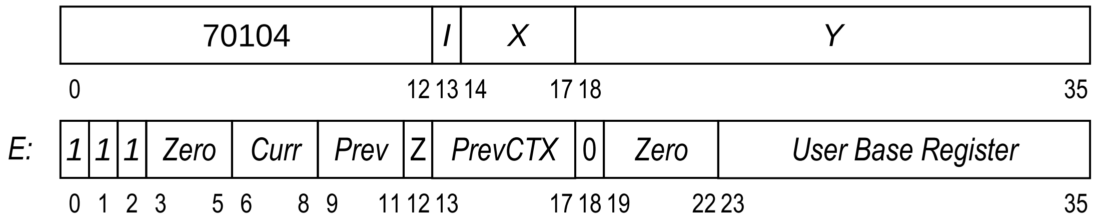
1.1.10 SETCU - Set CST-update-needed bits

70100	*I*	*X*	*Y*
0 12	13	14 17	18 35

Description:

Not implemented.

1.1.11 RDUBR – Read User Base Register



Description

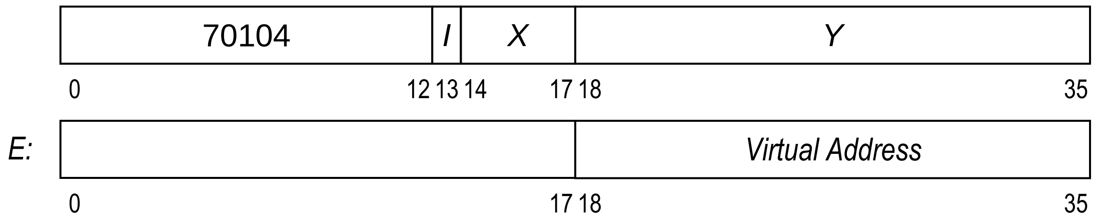
Bits:

Bit 0 - Read as 1
Bit 1 - Read as 1
Bit 2 - Read as 1
Bit 6-8 - Current AC Block
Bit 9-11- Previous AC Block
Bit 12 -
Bit 13-17- Previous Context Section
Bit 18 - Read as zero
Bit 23-35 – User Base Register

Compatibility:

Notes:

1.1.12 CLRPT – Clear Page Table Entry



Description

This instruction clears the hardware page table so that the next reference to the word at E will cause a refill cycle.

Bits:

Bit 18-35 – Virtual Address to Clear

Compatibility:

This instruction is compatible with the KS10.

Notes:

1.1.13 WRUBR – Write to the User Base Register

70114	*I*	*X*	*Y*
0 12	13	14 17	18 35

E:	0	1	2	*Zero*	*Curr*	*Prev*	*Z*	*Prev CTX*	*U*	*Zero*	*User Base Register*
	0	1	2	3 5	6 8	9 11	12	13 17	18	19 22	23 35

Description

This instruction loads user process context with the words at E:

Bits:

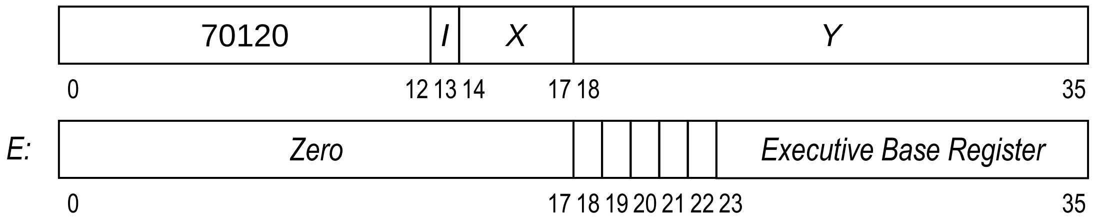
Bit 0 – Select the current and previous context AC blocks specified by bits 6-8 and 9-11 of E, respectively.
Bit 1 – Select previous context section specified by bit 13-17of E.
Bit 2 – Load User Base Register (UBR)
Bit 3 – Invalidate entire page table if bit 3 is reset; invalidate only non-keep entries in page table if bit 3 is set.
Bit 6-8 – Current AC block number
Bit 9-11 – Previous context AC block number.
Bit 13-17 – Previous context section number.
Bit 18 – Do not update user's accounts, loc 504-507 of UPT.
Bit 23-35 – User base Register.

Compatibility:

This instruction is identical to the KL10 WRUBR Instruction. The KS10 WRUBR doesn't implement bit 1, bit 18, or bit 23-24.

Notes:

1.1.14 WREBR – Write to the Exec Base Register



Description

Configure the pager according to the effective address E:

Bits:

Bit 18 – Cache look.
Bit 19 – Cache load.
Bit 20 –
Bit 21 – TOPS20 Paging (must be same as APRID – Bit 0)
Bit 22 – Enable Trap and Paging
Bit 23-35 – Executive Base Address

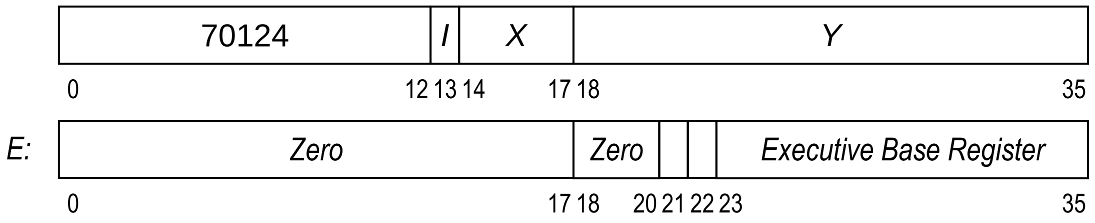
Compatibility:

This instruction is similar to the KL10 CONO PAG Instruction. Bits 18-20 control caching. Here they must be zero.

Notes:

It's also similar to KS10 WREBR except EBR is Bit 23-35 instead of Bit 25-35.

1.1.15 RDEBR – Read the Executive Base Register



Description

Bits:

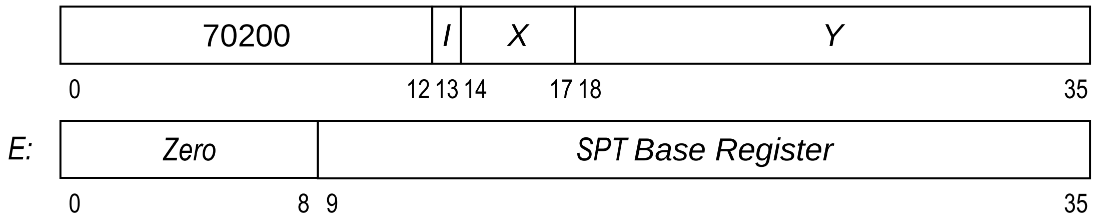
Bit 18-20 – Zero
Bit 21 – TOPS20 Paging (must be same as APRID – bit 0)
Bit 22 – Enable Trap and Paging
Bit 23-35 – Executive Base Address

Compatibility:

This instruction is similar to the KL10 CONI PAG Instruction. Bits 18-20 control caching. Here they must be zero. It's also similar to KS10 RDEBR except EBR is Bit 23-35 instead of Bit 25-35.

Notes:

1.1.16 RDSPB - Read SPT Base Register



Description

Read the contents of the SPT base register into bits 9-35 of location E.

Bits:

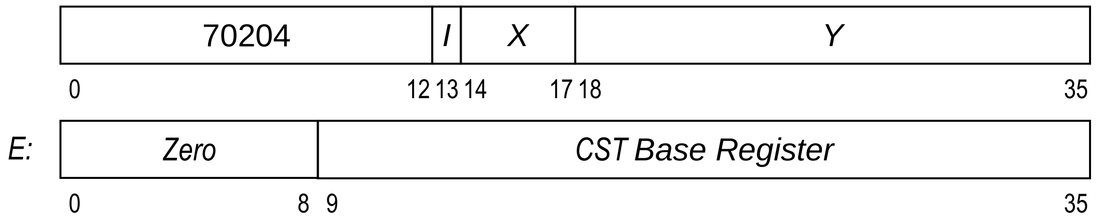
Bit 9-35 – SPT (Shared Pointer Table) Base Register

Compatibility:

The KS10 doc says SPT Base registers is bit 14-35

Notes:

1.1.17 RDCSB – Read CST Base Register



Description:

Read the contents of the of the CST base register into bits 9-35 of location E.

Bits:

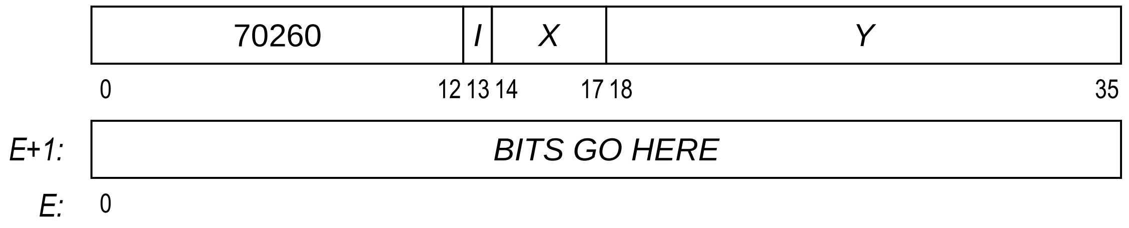
Bit 9-35 – CST (Core Status Table) Base Register

Compatibility:

The KS10 doc says SPT Base registers is bit 14-35

Notes:

1.1.18 RDPUR – Read Process Use Register



Description

Description

Bits:

Bits -

Compatibility:

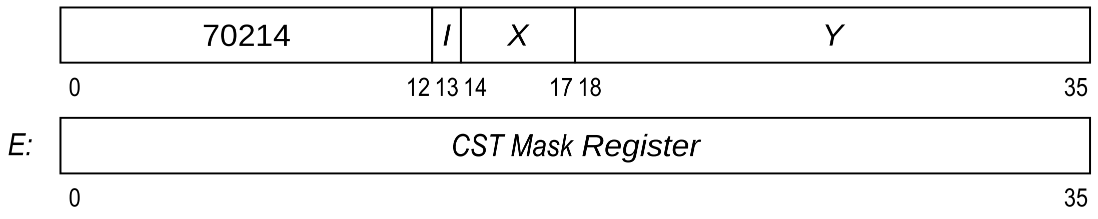
Compatibility

Notes:

Notes:

1.1.19 RDCSTM

Read CST Mask Register



Bit 0-35 – CST Mask Register

Description

Description

Bits:

Bits -

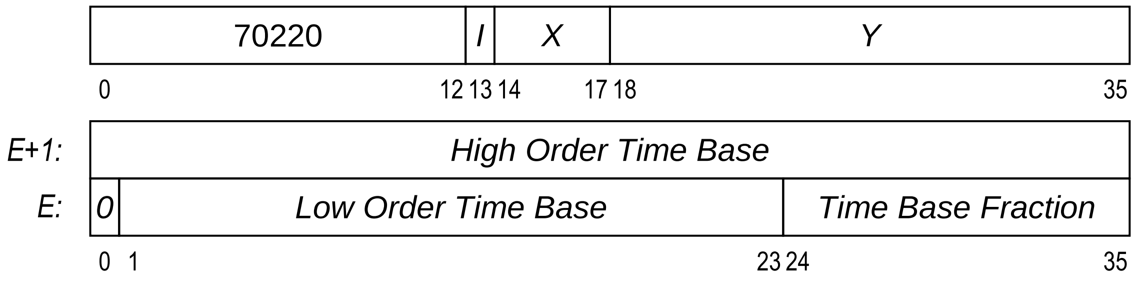
Compatibility:

Compatibility

Notes:

Notes:

1.1.20 RDTIM – Read Timebase condition



Description

Read the contents of the time base registers, add the current contents of the millisecond counter to the double-word read, and place the result in location E, E+1.

Bits:

See above.

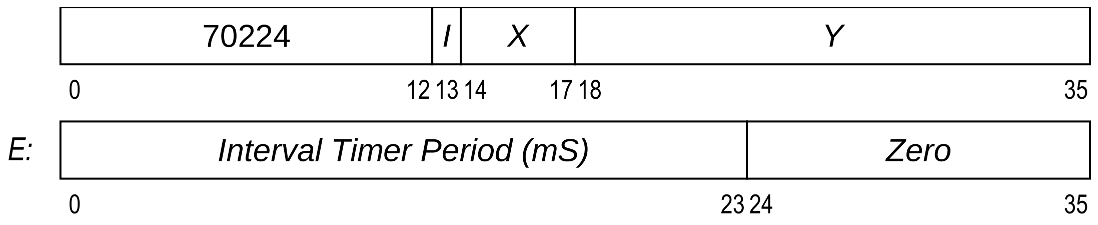
Compatibility:

This instruction is identical to the KS10 RDTIM Instruction.

Notes:

Because the hardware is set up for signed arithmetic, bit 0 of the lower order word must be skipped.

1.1.21 RDINT – Read Interval Timer conditions



Description

The RDINT instruction reads the current value of the Interval Timer Period Register and stores the value in E. The period read is the same as that was supplied by WRINT.

Bits:

Bit 0-23 – Interval Timer Period

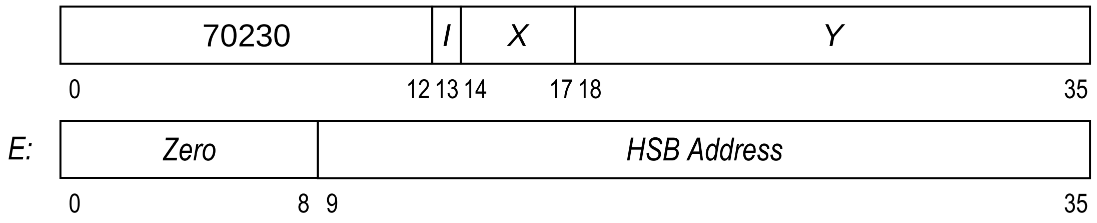
Compatibility:

This instruction is identical to the KS10 RDINT Instruction.

Notes:

None.

1.1.22 RDHSB - Read Halt Status Block address



Description

The RDHSB

Bits:

Bit

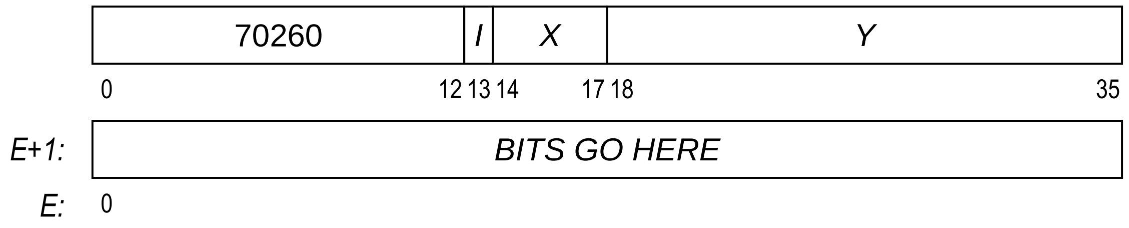
Compatibility:

This instruction is identical to the KS10 RDHSB Instruction.

Notes:

None.

1.1.23 WRSPB - Write SPT Base Register



<u>Description</u>

Description

<u>Bits:</u>

Bits -

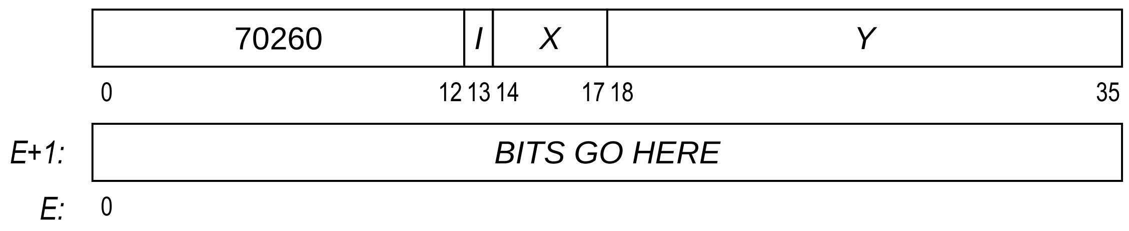
<u>Compatibility:</u>

Compatibility

<u>Notes:</u>

Notes:

1.1.24 WRCSB – Write CST Base Register



Description

Description

Bits:

Bits -

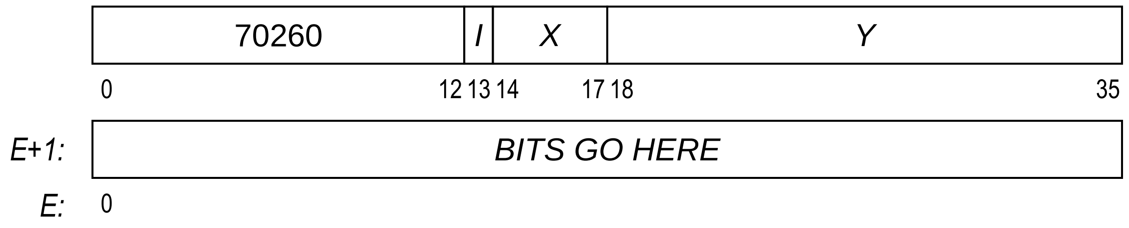
Compatibility:

Compatibility

Notes:

Notes:

1.1.25 WRPUR – Write Process Use Register



Description

Description

Bits:

Bits -

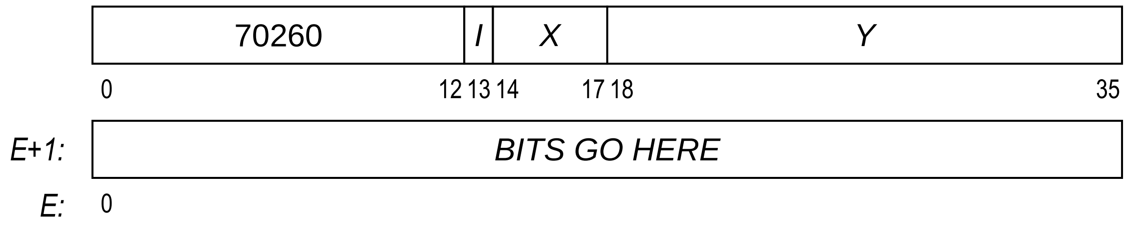
Compatibility:

Compatibility

Notes:

Notes:

1.1.26 WRCSTM - Write CST Mask Register



Description

Description

Bits:

Bits -

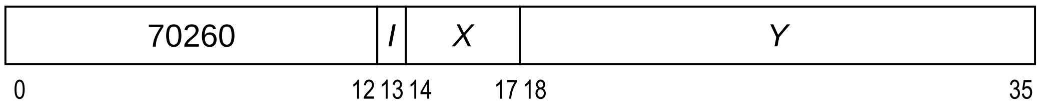
Compatibility:

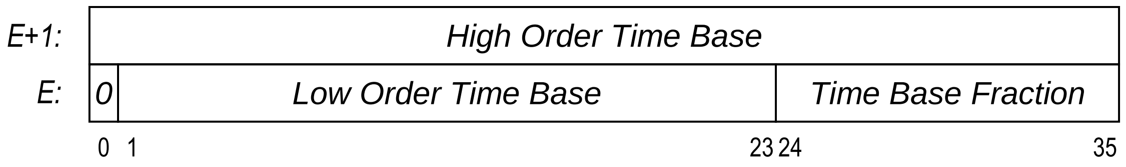
Compatibility

Notes:

Notes:

1.1.27 WRTIM - Write Timebase conditions





Description

Read the contents of location E,E+1, clear the right twelve bits of the low order word read (the part corresponding to the hardware millisecond counter), and place the result in the time base registers in the workspace.

Bits:

See above.

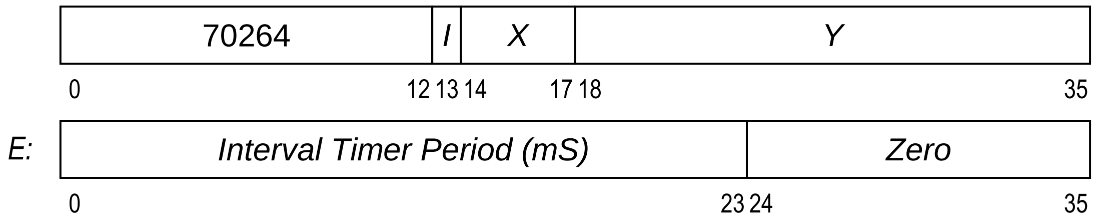
Compatibility:

This instruction is identical to the KS10 WRTIM Instruction.

Notes:

Because the hardware is set up for signed arithmetic, bit 0 of the lower order word must be skipped.

1.1.28 WRINT - Write Interval Timer conditions



Description

The WRINT instruction loads the Interval Timer Period Register and Interval Counter from E.

If the Interval Timer Period Register is set to zero, the Interval Timer function is disabled and the Period Counter does not decrement.

Bits:

Bit 0-23 – Interval Timer Period

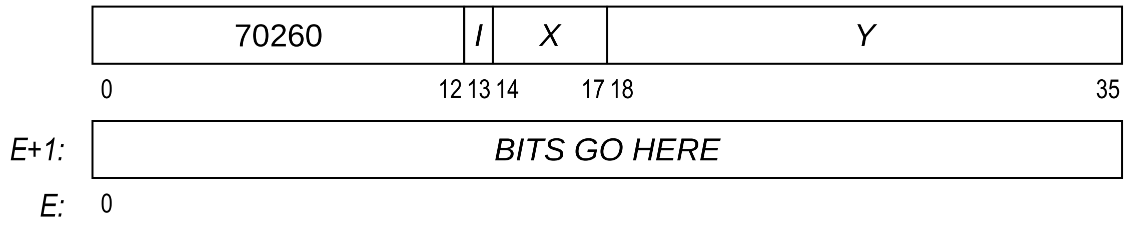
Compatibility:

This instruction is identical to the KS10 WRINT Instruction.

Notes:

None.

1.1.29 WRHSB - Write Halt Status Block Address



Description

Description

Bits:

Bits -

Compatibility:

Compatibility

Notes:

Notes:

2 Timers

2.1 Time Base

The Time Base is a 72-bit counter that increments at 4.096 MHz.

The top 60 bits represent time in milliseconds and the lower 12-bits (if implemented) represent fractional milliseconds.

2**12 = 4096; 4.096 MHz / 4096 = 1 KHz.

2.1.1 Minnow Time Base

The comments in the minnow microcode, and KC10 documents, state (incorrectly, I think) that the time base is in microseconds. More correctly it is in 1/4.096 uS per LSB. Furthermore the KF10 (Dolphin) Design Documents[i] state that the KS10

The microcode handles a 27000 Hz interrupt from the timer hardware.

The microcode increments the time base by 4000 every 27 timer ticks (one millisecond). I'm guessing that the time base really should increment by 4096 every millisecond.

Here's where things don't make any sense:
The clock for the hardware runs at 20.277 MHz. A counter divides the clock by 2167. That's nowhere near 27KHz. A divider of 751 would be exactly 27000 KHz.

Furthermore, why would you want to run the timer interrupt at 27 KHz? It just wastes micro-instruction cycles. Was Minnow so cost sensitive that they could afford a 12-bit counter and not a 16-bit counter that could generate 1 millisecond interrupts directly?

2.2 Interval Timer Operation

The Interval Timer is used to generate a periodic interrupt. It consists of a Interval Timer Period Register and a preset-able Interval Counter. When the WRINT instruction is called, the both the Interval Timer Period Register and Interval Counter are set to E.

Every millisecond, the Interval Counter is decremented an compared to zero. If the Interval Counter is zero, an Interval Timer Done bit (RDAPR bit 30) is asserted, and the Interval Counter is re-loaded with the value of the Interval Timer Period Register.

2.2.1 Minnow Interval Timer

The interval timer is implemented completely in microcode. There is no interval timer hardware.

The Interval Counter is located at address 565 in the RAMFILE; the Interval Timer Period Register (Time Interval) is located at address 564 in the RAMFILE.

3 ALU

The PDP-10 ALU supports Fixed-Point and Floating-Point arithmetic in both single-precision and double-precision formats.

3.1 Single Precision Fixed Point Arithmetic

ADDx, SUBx, IMULx, IDIVx, MULx, DIVx

LSH, LSHC, ASH, ASHC, ROT, ROTC, JFFO

SETZ, AND, ANDCA, SETM, ANDCM, SETA, XOR, IOR, ANDCB, EQV, SETCA, ORCA, SETCM, ORCM, ORCB, SETO.

3.2 Double Precision Fixed Point Arithmetic

DADD, DSUB, DMUL, DDIV

3.3 Single Precision Floating Point Arithmetic

FADx, FADRx, FSBx, FSBRx, FMPx, FMPRx, FDVx, FDVRx, FSC, UFA, FIX, FIXR, FLTR

3.4 Double Precision Floating Point Arithmetic

DFAsDxx, DFSB, DFMP, DFDV, DFN

4 Pager

5 Priority Interrupt Controller

6 Memory

7 UART I/O (Local Bus)

The Minnow Platform support up to 8 UARTs for communicating with 8 RS-232 terminals. These UARTS are isolated to an 8-bit little-endian (bit-order reversed) bus.

Whereas the original Minnow design uses a tri-state bus for the 8 UARTS, the UART designs have been modified to provide a separate input and output bus. The UART output buses are multiplexed into a single output bus for this block. These changes make the design more compatible with normal FPGA design practices.

The microcode and Phillips UART manual refer to the devices as the Programmable Communications Interface (PCI) – this is a bit unfortunate because abbreviation PCI has become well known for another use.

A block diagram of the UART I/O Block is illustrated below in **Figure 7-1**.

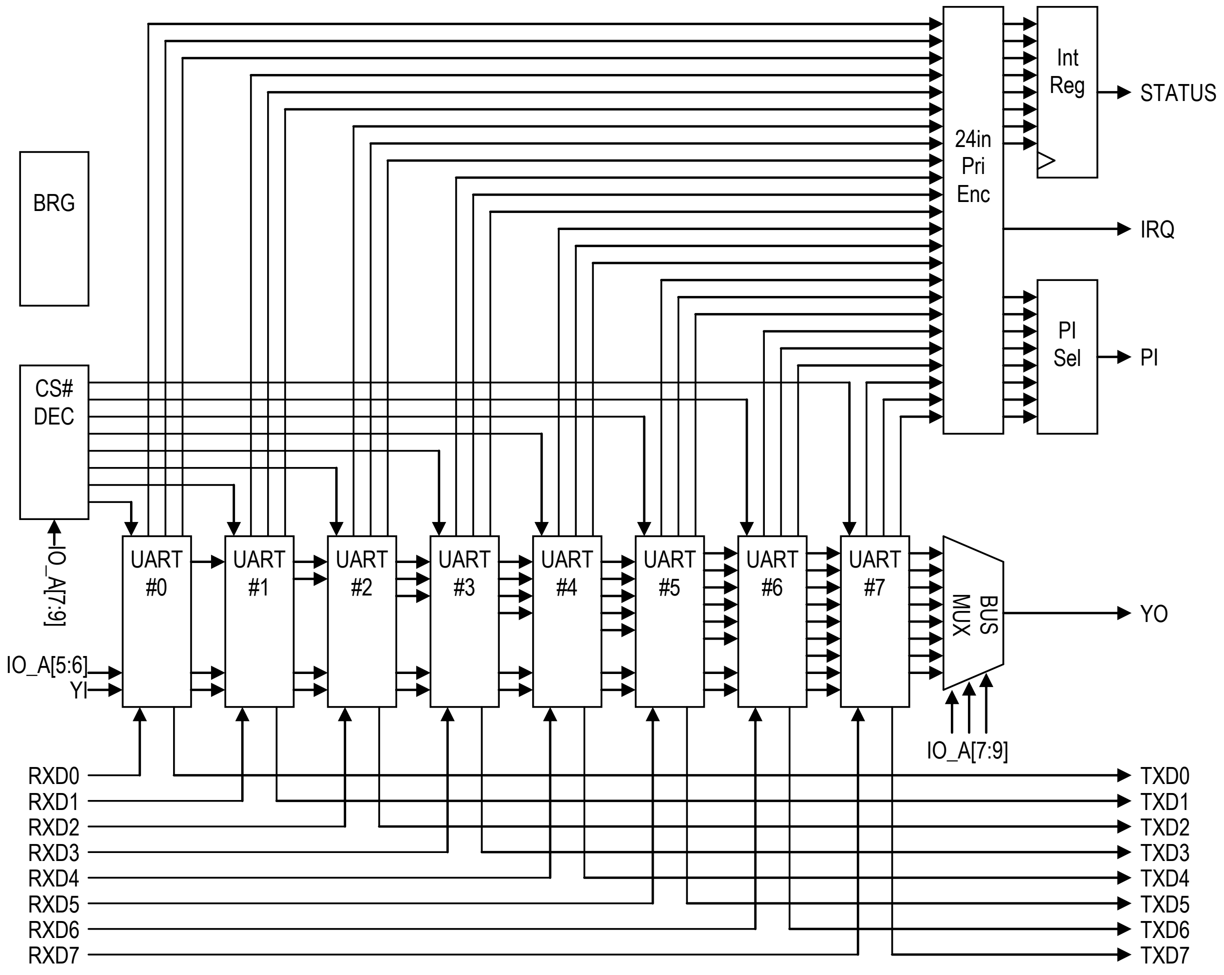


Figure 7-1 – UARTIO Block Diagram

The Minnow UARTIO subsystem implements a few design features that will not be re-implemented in the FPGA version. In the past 30 years, the use of RS-232 I/O has matured and history has shown that some of these features are not very useful.

These changes to the Minnow UARTIO subsystem are:

a. The Minnow UARTIO subsystem has the ability to use an external clock for both the UART Receivers and UART Transmitters. External clock selection is handled by a special register, MR2, that is implemented by MIN8:E719 and MIN8:E720. The ability to use external clocks for RS-232 IO will not be implemented in this design.
b. The Minnow UARTIO subsystem implements full-on RS-232 hardware flow control (DSR, DCD, CTS, RTS, DTR) even though the microcode implements XON/XOFF software flow control. Hardware flow control will not be implemented in this design.
c. I don't think we'll be needing 8 UARTs. Ethernet maybe.
d. Various unused features of the SCN2651 UART will not be implemented.

7.1 UART I/O Memory Map

Table 1 - Memory Map

WR	IO_A[5:6]	IO_A[7:9]	Device	Register
1	11	111	UART7	Command Register
0	11	111	UART7	Command Register
1	10	111	UART7	Mode Register (not implemented)
0	10	111	UART7	Mode Register (not implemented)
1	01	111	UART7	SYN Register (not implemented)
0	01	111	UART7	Status Register (read only)
1	00	111	UART7	Transmit Data
0	00	111	UART7	Receive Data
1	11	110	UART6	Command Register
0	11	110	UART6	Command Register
1	10	110	UART6	Mode Register (not implemented)
0	10	110	UART6	Mode Register (not implemented)
1	01	110	UART6	SYN Register (not implemented)
0	01	110	UART6	Status Register (read only)
1	00	110	UART6	Transmit Data
0	00	110	UART6	Receive Data
1	11	101	UART5	Command Register
0	11	101	UART5	Command Register
1	10	101	UART5	Mode Register (not implemented)
0	10	101	UART5	Mode Register (not implemented)
1	01	101	UART5	SYN Register (not implemented)
0	01	101	UART5	Status Register (read only)
1	00	101	UART5	Transmit Data
0	00	101	UART5	Receive Data
1	11	100	UART4	Command Register
0	11	100	UART4	Command Register
1	10	100	UART4	Mode Register (not implemented)
0	10	100	UART4	Mode Register (not implemented)
1	01	100	UART4	SYN Register (not implemented)
0	01	100	UART4	Status Register (read only)
1	00	100	UART4	Transmit Data
0	00	100	UART4	Receive Data

Table 1 - Memory Map (continued)				
WR	**IO_A[5:6]**	**IO_A[7:9]**	**Device**	**Register**
1	11	011	UART3	Command Register
0	11	011	UART3	Command Register
1	10	011	UART3	Mode Register (not implemented)
0	10	011	UART3	Mode Register (not implemented)
1	01	011	UART3	SYN Register (not implemented)
0	01	011	UART3	Status Register (read only)
1	00	011	UART3	Transmit Data
0	00	011	UART3	Receive Data
1	11	010	UART2	Command Register
0	11	010	UART2	Command Register
1	10	010	UART2	Mode Register (not implemented)
0	10	010	UART2	Mode Register (not implemented)
1	01	010	UART2	SYN Register (not implemented)
0	01	010	UART2	Status Register (read only)
1	00	010	UART2	Transmit Data
0	00	010	UART2	Receive Data
1	11	001	UART1	Command Register
0	11	001	UART1	Command Register
1	10	001	UART1	Mode Register (not implemented)
0	10	001	UART1	Mode Register (not implemented)
1	01	001	UART1	SYN Register (not implemented)
0	01	001	UART1	Status Register (read only)
1	00	001	UART1	Transmit Data
0	00	001	UART1	Receive Data
1	11	000	UART0	Command Register
0	11	000	UART0	Command Register
1	10	000	UART0	Mode Register (not implemented)
0	10	000	UART0	Mode Register (not implemented)
1	01	000	UART0	SYN Register (not implemented)
0	01	000	UART0	Status Register (read only)
1	00	000	UART0	Transmit Data
0	00	000	UART0	Receive Data

7.2 UART Interrupt Priority Encoder and Status Register

The Interrupt Priority Encoder Tree looks at all 24 of the UART interrupts, determines if a interrupt is active, determines the priority of the current interrupt (if active) and feeds the two sets of signals to the interrupt controller. Because multiple interrupt sources share the same interrupt, a status register that can be queried by the microcode is provided. **Error! Reference source not ound.** summarizes the operation of the priority encoder.

Table 2 – Status Register

0	**IRQ**	**0**	**TYPE**		**IRQ**			**Description**	**Priority**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**		
0	1	0	1	0	1	1	0	DSCHG 7	23 (highest)
0	1	0	1	0	1	1	0	DSCHG 6	22
0	1	0	1	0	1	0	1	DSCHG 5	21
0	1	0	1	0	1	0	0	DSCHG 4	20
0	1	0	1	0	0	1	1	DSCHG 3	19
0	1	0	1	0	0	1	0	DSCHG 2	18
0	1	0	1	0	0	0	1	DSCHG 1	17
0	1	0	1	0	0	0	0	DSCHG 0	16
0	1	0	0	1	1	1	0	TXRDY 7	15
0	1	0	0	1	1	1	0	TXRDY 6	14
0	1	0	0	1	1	0	1	TXRDY 5	13
0	1	0	0	1	1	0	0	TXRDY 4	12
0	1	0	0	1	0	1	1	TXRDY 3	11
0	1	0	0	1	0	1	0	TXRDY 2	10
0	1	0	0	1	0	0	1	TXRDY 1	9
0	1	0	0	1	0	0	0	TXRDY 0	8
0	1	0	0	0	1	1	0	RXRDY 7	7
0	1	0	0	0	1	1	0	RXRDY 6	6
0	1	0	0	0	1	0	1	RXRDY 5	5
0	1	0	0	0	1	0	0	RXRDY 4	4
0	1	0	0	0	0	1	1	RXRDY 3	3
0	1	0	0	0	0	1	0	RXRDY 2	2
0	1	0	0	0	0	0	1	RXRDY 1	1
0	1	0	0	0	0	0	0	RXRDY 0	0
0	0	0	0	0	0	0	0	Nothing Active	(lowest)

The Status Register illustrate in **Error! Reference source not found.** above is an 8-bit register.

1. Bit-0 (MSB) is not used and is always zero.
2. Bit-1 (IRQ) is asserted any time any of the interrupts are asserted.
3. Bit-2 is not used and is always zero.
4. Bit-3 through Bit-4 are identified as follows:
    a. 00 – RX Interrupt
    b. 01 – TX Interrupt
    c. 10 – DSCHG Interrupt
    d. 11 – Not encoded
5. Bit-5 though Bit-7 (LSB) are identified as follows:
    a. 000 – UART 0
    b. 001 – UART 1
    c. 010 – UART 2
    d. 011 – UART 3
    e. 100 – UART 4
    f. 101 – UART 5
    g. 110 – UART 6
    h. 111 – UART 7

7.3 UART Baud Rate Generator

The Baud Rate Generator (BRG) is responsible for generating a 1X clock for the UART transmitters and a 16X clock for the UART receiver.

7.4 SCN2651 UART Design

The SCN2651 UART most likely was chosen for the Minnow design because it can support separate RX, TX, and Status Change interrupts. As we've already discusses, all three interrupts are used by the Minnow microcode to buffer input and output streams.

In this design only the features of the SCN2651 UART that are used by the Minnow microcode are implemented. Arguably, it is just as easy to change the UART VHDL Code as it is to change the microcode.

Synchronous UART Modes are not implemented.

Asynchronous UART settings are hard-wired and not changeable.

7.4.1 SCN2651 Register Set

The register set of a full implementation of the SCN2651 is illustrated below in Table 3. This UART implementation backward compatible

Table 3 – SCN2651 Register Addressing

CE#	A_1	A_0	R#/W	Function	Comment
1	X	X	X	Tristate data bus	
0	0	0	0	Read receive holding register	
0	0	0	1	Write transmit holding register	
0	0	1	0	Read status register	
0	0	1	1	Write SYN1/SYN2/DLE register	Not implemented
0	1	0	0	Read Mode Register	Reads constant data
0	1	0	1	Write Mode Register	Not implemented
0	1	1	0	Read Command Register	
0	1	1	1	Write Command Register	

7.4.1.1SCN2651 Receive Holding Register

The UART Receiver is double buffered, The Receiver Holding Register provides one level of buffer and the UART Receiver shift register provides another buffer.

Data is received when the CMD.TxEN bit is asserted.

When the Receiver Holding Register contains received data, the STAT.RxRDY bit is asserted and RxRDY pin is asserted.

7.4.1.2SCN2651 Transmit Holding Register

The UART Transmitter is also double buffered. The Transmit Holding Register provides one level buffer and the UART Transmitter shift register provides another buffer.

Data is transmitted when the CMD.TxEN bit is asserted.

When the Transmitter Holding Register contains nothing to transmit, the STAT.TxRDY bit is asserted and TxRDY pin is asserted.

7.4.1.3SCN2651 Status Register (Read Only)

The status register

Table 4 – SCN2651 Status Register (Read Only)

SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
Data Set Ready	**Data Carrier Detect**	**FE Detect**	**Overrun**	**PE**	**DSCHG**	**RxRDY**	**TxRDY**
Read as 0	Read as 0	0 = Normal 1 = Framing	0 = Normal 1 = Overrun error	0 = Normal 1 = Parity error	Read as 0	0=Receive holding register empty 1=Receive holding register has data	0=Transmit holding register busy 1=Transmit holding register empty

SCN2651 SYN1/ This register is not implemented. Writes to this register are ignored.

7.4.1.4SCN2651 SYN2/DLE Register (Write Only)

7.4.1.5SCN2651 Mode Register 1 (Read Only)

The mode register (unlike a regular SCN2651 is Read Only. Writes to this register are ignored.

The UART is pre-configure for 9600 baud, no parity, 8 bits of data, 1 stop bit with an internal 16x clock rate. The UART configuration cannot be changed.

Table 5 – SCN2651 Mode 1 Register (Read Only)

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
Stop bit length		**Parity Type**	**Parity Control**	**Character Length**		**Mode and Baud Rate Factor**	
Read as 01		Read as 0	Read as 0	Read as 11		Read as 10	

7.4.1.6SCN2651 Mode Register 2 (Read Only)

The mode register (unlike a regular SCN2651 is Read Only. Writes to this register are ignored.

Table 6 – SCN2651 Mode 2 Register (Read Only)

MR27	MR26	MR25	MR24	MR23	CR22	CR21	CR20
Not used		**Transmitter Clock**	**Receiver Clock**	**Baud Rate Selection**			
Read as 00		Read as 1	Read as1	Read as 1111			

7.4.1.7SCN2651 Command Register (Read/Write)

Only the RxEN and TxEN bits of the Command Register are implemented. The other bits are ignored on writes and read-back as zero.

Table 7 – SCN2651 Command Register (Read/Write)

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
Operating Mode		Request to Send	Reset Error	Break	Receive Control (RxEN)	Data Terminal Ready	Transmit Control (TxEN)
Read as 00		Read as 0	Read as 0	Read as 0	0 = Disable 1 = Enable	Read as 0	0 = Disable 1 = Enable
Writes Ignored		Writes Ignored	Writes Ignored	Writes Ignored		Writes Ignored	

8 Microsequencer

The microsequencer generates the addresses used to step through the microprogram of the control store (microcode).

A block diagram of the modified Minnow microsequencer is illustrated below in Figure 5-**8**-**1**.

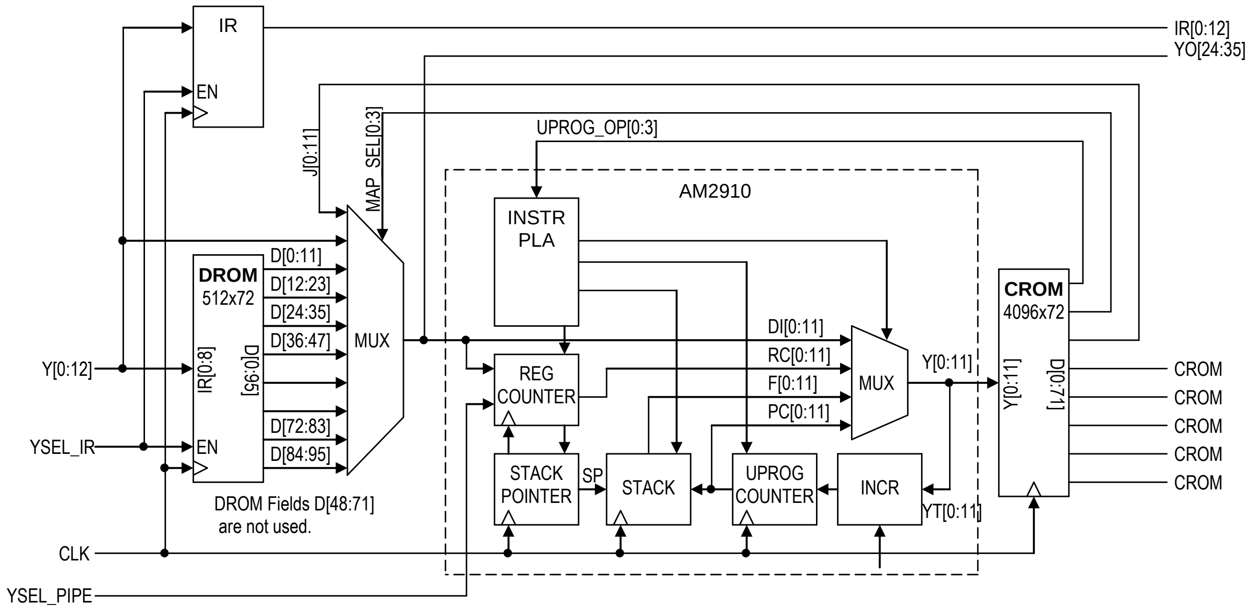


Figure 5-8-1 - Microsequencer Block Diagram

This implementation of this microsequencer is different than the Minnow implementation as follows:

1. The Dispatch ROM or DROM has been re-organized from a 4096 x 12 ROM into a 512 x 96 ROM. This allows the design to be fully synchronous.

2. The Dispatch ROM or DROM has been changed from a normal static ROM to a fully synchronous ROM which is compatible with FPGA-based ROM.

3. All tri-state buses have been replaced with buses and multiplexers.

4. The microsequencer stack size has been increased from 5 levels to 7 levels and could be trivially made larger. Why? Because I could.

Although these design changes affect the block diagram, the operation of the microsequencer is unchanged from the original design. It maintains strict microcode compatibility with the Minnow design.

The microsequencer executes in single clock cycle. No pipelines are required. A preliminary "place and route" estimate indicates that the microsequencer will execute at about 125 MHz in a Xilinx XC6SLX16 FPGA.

8.1 Dispatch ROM

The Dispatch ROM provides an interface between the PDP-10 instruction set and the underlying microsequencer. It accomplishes this by using the PDP-10 Opcode as an address into the Dispatch ROM. For each opcode (and associated address) the Dispatch ROM provides a Control ROM address than implements that instruction. In other words, when an opcode is decoded, the Dispatch ROM supplies the address in the microcode to begin executing that instruction.

The "Minnow" Dispatch ROM is a 4096 (12-bit) by 12-bit ROM. The 3 LSB of address are controlled by the Control ROM and the upper 9 MSBs are controlled by the PDP-10 Opcode.

The contents of the ROM are read by making eight accesses to the ROM. A better way to view the Dispatch ROM would be as a 512 (9-bit) by 96-bit ROM. For each of the 512 PDP-10 Opcodes there are up to eight 12-bit fields that point into microcode that describe how to decode the instruction.

At present, two of the 12-bit fields are unused on every instruction. Some ROM savings in the FPGA may be obtained by using a 512 by 72-bit Dispatch ROM. However, the VHDL optimizer is smart and figures out that the those two12-bit ROM fields are constant and replaces the ROM fields with a constant.

A second design change to the original Minnow design is required so that the Dispatch ROM may be a implemented as a registered ROM. In the original design, the y-bus clocked data into the Instruction Register (IR) and the op-code portion of the Instruction Register was used as the address of the Dispatch ROM. In the original implementation the Dispatch ROM is unclocked, the Instruction Register provides the Opcode “memory”. In the revised, FPGA compatible implementation, the Dispatch ROM and the Instruction Register are loaded synchronously and simultaneously. This saves a clock cycle that would have been necessary in order to clock the Instruction Register into the Dispatch ROM

The various Dispatch ROM fields are used to simplify the decoding of similar instructions.

For example (reading the microcode) the "HRRZM" instruction (opcode 552) uses four microcode "entry points" to decode that instruction:

```
    "FETCH AC", "HRRZX", "MCE, "TO MEM"
```

The HRRZS insruction is decoded similarly:

```
    "FETCH MEM(W)", "HRRZX", "MCE", "BACK TO SELF"
```

This approach allows the "HRRZX" microcode to be used for more than one instruction.

I suspect that "Minnow" is implemented like it is because it uses fewer, larger ROMS.

8.2 Am2910 Microsequencer

This section describes the Am2910 Microprogram Controller. Quoting the user manual "This device is an address sequencer intended for controlling the sequence of execution of microinstructions stored in microprogram memory”. Besides the capability of sequential access, it provides conditional branching to any microinstruction within its 4096-microword range. A last-in, first-out stack provides microsubroutine return linkage and looping capability; there are five levels of nesting of microsubroutines. Microinstruction loop count control is provided with a count capacity of 4096.

During each microinstruction, the Microprogram controller provides a 12-bit address from one of four sources:

a. the microprogram address register (PC) which usually contains an address one greater than the previous address;
b. an external (direct) input (D); or
c. a register/counter (R) retaining data loaded during a previous microinstruction; or
d. a seven deep last-in, first-out stack (F)."

8.2.1 Am2910 Condition Code Selection.

Most microsequencer operations are conditional. The "`TEST SEL`" microcode field selects the condition code for the microsequencer. Asserting the value of "`ALWAYS`" onto the "`TEST SEL`" microcode field may be will perform an unconditional operation.

Additionally the Am2904 Carry Shifter status output is connected to the "`CT`" input of the "`TEST SEL`" multiplexer. Therefore the ALU status may be tested by asserting the value of "`CT`" or "`NOT CT`" onto the "`TEST SEL/`" microcode field and asserting proper microcode to Am2904 ALU Carry Shifter.

8.2.2 Am2910 Jumps

Refer to Figure 5-**8-1** for a description of the various buses and multiplexers.

For the purpose of this discussion, a goto, jump, and branch all perform the same type of operation. The microcode uses the dreaded goto terminology.

Computed jumps are performed by asserting the value of '1' on the "`ALU_Y`" microcode field and asserting the value of "`PIPELINE`" on the "`Y SEL`" microcode field. This connects the "Y Bus" input (which contains the last result of the last ALU operation) to the "D Bus" via the big multiplexer. Lastly the `CJP` operation is asserted onto the "`U PROG OP`" microcode field.

Immediate jumps are performed by selecting the "J" microcode field onto the "D" Bus by asserting the value of '0' on the "`ALU_Y`" microcode field or asserting any value other than "`PIPELINE`" on the "`Y SEL`" microcode field. (I.e., not a computed jump). Here the `CJP` operation is asserted onto the "`U PROG OP`" microcode field.

The following microcode macros provide the various conditional and unconditional jump operations.

Microcode Macro Name	Description	Operation
`GOTO Y`	Unconditional Computed Jump	Unconditionally jump to the address that is the result of the last ALU operation (12 most MSBs).
`GOTO []`	Unconditional Immediate Jump	Unconditionally jump to the address that is supplied by the J microcode field.
`GOTOP []`	Unconditional Immediate Jump, POP	Unconditionally jump to the address that is supplied by the J microcode field and pop stack.
`IF [] THEN Y`	Conditional Computed Jump	Conditionally jump to the address that is the result of the last ALU operation (12 most MSBs) using the TEST SEL multiplexer as the condition.

Microcode Macro Name	Description	Operation
`IF [] THEN []`	Conditional Immediate Jump	Conditionally jump to the address that is supplied by the J microcode field using the TEST SEL multiplexer as the condition.
`IF [] THENP []`	Conditional Immediate Jump, POP	Conditionally jump to the address that is supplied by the J microcode field using the TEST SEL multiplexer as the condition.
`IF CT [] THEN Y`	Conditional Computed Jump ALU Status	Conditionally jump to the address that is the result of the last ALU operation (12 most MSBs) using the Am2904 Carry/Shifter ALU Status as the condition.
`IF CT [] THEN []`	Conditional Immediate Jump ALU Status	Conditionally jump to the address that is supplied by the J microcode field using the Am2904 Carry/Shifter ALU Status as the condition
`IF CT [] THENP []`	Conditional Immediate Jump, POP ALU Status	Conditionally jump to the address that is supplied by the J microcode field using the Am2904 Carry/Shifter ALU Status as the condition with POP.
`IF NOT CT [] THEN Y`	Conditional Computed Jump ALU Status	Conditionally jump to the address that is the result of the last ALU operation (12 most MSBs) using the negated Am2904 Carry/Shifter ALU Status as the condition.
`IF NOT CT [] THEN []`	Conditional Immediate Jump ALU Status	Conditionally jump to the address that is supplied by the J microcode field using the negated Am2904 Carry/Shifter ALU Status as the condition
`IF NOT CT [] THENP []`	Conditional Immediate Jump, POP	Conditionally jump to the address that is supplied by the J microcode field using the negated Am2904 Carry/Shifter ALU Status as the condition with POP.

8.2.3 Am2910 Conditional Calls

Computed calls and Immediate calls are programmed the same as Jump instruction described in Section 8.2.2 except that a `CJS` operation is used. This instruction pushes a return address onto the stack so that a return instruction can return the execution to the next statement.

Microcode Macro Name	Description	Operation
`CALL Y`	Unconditional Computed Call	Unconditionally call the address that is the result of the last ALU operation (12 most MSBs). Save the return address on the stack.
`CALL []`	Unconditional Immediate Call	Unconditionally call the address that is supplied by the J microcode field. Save the return address on the stack.
`IF [] CALL []`	Conditional Immediate Call	Conditionally call to the address that is supplied by the J microcode field using the TEST SEL multiplexer as the condition. Save the return address on the stack.
`IF CT [] CALL []`	Conditional Immediate Call	Conditionally call to the address that is supplied by the J microcode field using the Am2904 Carry/Shifter ALU Status as the condition. Save the return address on the stack.
`IF NOT CT [] CALL []`	Conditional Immediate Call	Conditionally call to the address that is supplied by the J microcode field using the negated Am2904 Carry/Shifter ALU Status as the condition. Save the return address on the stack.

8.2.4 Am2910 Dispatch Operations

Refer to Figure 5-**8-1** for a description of the various buses and multiplexers.

Dispatch operations are performed by loading a instruction into the Instruction Register (`IR`) and Dispatch ROM. This is accomplished by reading the `IR` from the ALU, asserting 1 onto the “`ALU_Y`” microcode field, and asserting the value “`LOAD IR`” onto the “`Y SEL`” microcode field.

The eight possible dispatch vectors are selected by asserting various values onto the “`MAP SEL`” microcode field as well as by using the `VECT` output of the microsequencer which is only asserted during a `CJV` operation.

Lastly a `CJV` operation is asserted onto the “`U PROG OP`” microcode field.

8.2.5 Am2910 Conditional Returns

Microcode Macro Name	Description	Operation
`RETURN`	Unconditional Return	Unconditionally return to address on the stack.
`IF [] RETURN`	Conditional Return	Conditionally return to the address on the stack using the TEST SEL multiplexer as the condition.
`IF CT [] RETURN`	Conditional Return	Conditionally return to the address on the stack using the Am2904 Carry/Shifter ALU Status as the condition.
`IF NOT CT [] RETURN`	Conditional Return	Conditionally return to the address on the stack using the negated Am2904 Carry/Shifter ALU Status as the condition.

8.2.6 Am2910 Loops

Loops are formed using two microinstructions: the PUSH instruction which initializes the stack and stores immediate data in a counter and the RFCT instruction or a

8.3 Control ROM

The control ROM contains the control store (or microcode) and is configured as a 4K x 72 bit synchronous ROM. The Control ROM is organized as follows:

Low Bit	**High Bit**	**Name**	**Description**
0	8	ALU	am2903 ALU OP
9	14	A_SEL	am2903 ALU Reg A-port Addr (read only)
15	20	B_SEL	am2903 ALU Reg B-port Addr (read/write)
21	22	CARRY_OP	
23	28	SHARED_OP	Various shared purposes
29	34	STATUS_OP	
35	35	M_STATUS_ENB	
36	36	U_STATUS_ENB	
37	37	ALU_Y	ALU is driving the Y Bus.
38	41	U_PROG_OP	am2910 Microsequencer OP
42	47	TEST_SEL	Test Select Multiplexer
48	59	J	Immediate Data
60	62	MEM_OP	
63	66	Y_SEL	Y multiplexer control
67	69	T	Not used
70	70	MARK1	Not used
71	71	MARK2	Not used

9 RAMFILE

The RAMFILE is a 1k x 36 chunk of memory that is used by the microcode. It contains a scratch pad area that is accessible by the microcode, an area that contains the eight AC blocks that is addressable by the microcode and is specially addressable using the AC Block and current instruction, and the Page Table.

Address (octal)	Description
0000 – 0577	Microcode Scratch Pad. See microcode listing (J field) for contents.
0600 – 0617	AC Block 0, R0-R17
0620 – 0637	AC Block 1
0640 – 0657	AC Block 2
0660 – 0677	AC Block 3
0700 – 0717	AC Block 4
0720 – 0737	AC Block 5
0740 – 0757	AC Block 6
0760 – 0777	AC Block 7
1000 – 1777	Page Table.

9.1 RAMFILE Address Modes

The various special addressing modes for the RAMFILE are controlled by the SPEC SEL/ microcode field. The addressing into the RAMFILE is as follows:

SPEC SEL/ 0123_45	Address Mode	A_0 MSB	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9 LSB	Comment
0000_11	Direct	Y_{26}	Y_{27}	Y_{28}	Y_{29}	Y_{30}	Y_{31}	Y_{32}	Y_{33}	Y_{34}	Y_{35}	
0100_11	AC Block | VMA	0	1	1	B_o	B_1	B_2	Y_{32}	Y_{33}	Y_{34}	Y_{35}	
0101_11	AC Block | AC+N	0	1	1	B_o	B_1	B_2	A_0	A_1	A_2	A_3	
0110_11	AC Block | XR	0	1	1	B_o	B_1	B_2	X_0	X_1	X_2	X_3	
1010_ 11	AC Block | IW	0	1	1	B_o	B_1	B_2	X_0	X_1	X_2	X_3	
1111_11	Page Table	1	Y_{18}			Y_{21}	Y_{22}	Y_{23}	Y_{24}	Y_{25}	Y_{26}	
1111_11	Page Table	1	Y_{18}			Y_{21}	Y_{22}	Y_{23}	Y_{24}	Y_{25}	Y_{26}	
1111_11	Page Table	1	Y_{18}			Y_{21}	Y_{22}	Y_{23}	Y_{24}	Y_{25}	Y_{26}	
1111_11	Page Table	1	Y_{18}			Y_{21}	Y_{22}	Y_{23}	Y_{24}	Y_{25}	Y_{26}	

Where:

Y_{26} to Y_{35} are the lowest bits of the Y bus.
B_0 to B_2 are the current AC Block
A_0 to A_3 is AC of the current instruction plus offset for J.AC microcode field.
X_0 to X_3 is XR of the current instruction

9.1.1 Ramfile in Direct Access Mode

This RAMFILE mode is selected when the value of the symbol “`Y`” is asserted onto the “`SPEC SEL/`” microcode field.

The microengine uses the RAMFILE in Direct Access mode to store and retrieve data for it’s own purposes. The “`J/`” microcode field can be used to provide an immediate address in direct in direct mode. Therefore in this mode, the microengine is aware of the memory map therefore the locations of all the variables in the RAMFILE.

9.1.2 Ramfile in AC Block | VMA Mode

This RAMFILE mode is selected when the value of the symbol “`VMA`” is asserted onto the “`SPEC SEL/`” microcode field.

9.1.3 Ramfile in AC Block | AC+N Mode

This RAMFILE mode is selected when the value of the symbol “`AC+N`” is asserted onto the “`SPEC SEL/`” microcode field.

9.1.4 Ramfile in AC Block | XR Mode

This RAMFILE mode is selected when the value of the symbol “`XR`” is asserted onto the “`SPEC SEL/`” microcode field.

9.1.5 Ramfile in AC Block | IW Mode

This RAMFILE mode is selected when the value of the symbol “`IW`” is asserted onto the “`SPEC SEL/`” microcode field.

9.1.6 Ramfile in Page Table Mode

This RAMFILE mode is selected when the value of the symbol “`PAGE TABLE ENTRY`” is asserted onto the “`SPEC SEL/`” microcode field.

The Y bus has a virtual address in the following format:

Ignored	Section	*Page*	*Word in Page*
0 5	6 17	18 26	27 35

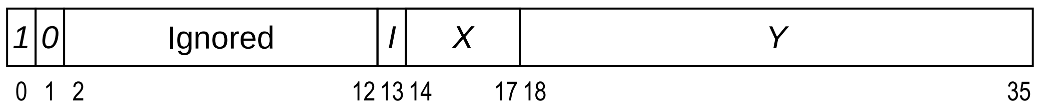


Figure 6-9-1 - Local Indirect Word

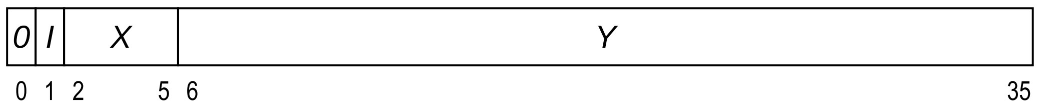


Figure 6-9-2 - Global Indirect Word

A local index is an 18-bit displacement or address obtained from an index register used in an effective address calculation in section zero, or from an index register used in a non-zero section that has bit 0=1 or bits 6-17 equal zero. In a non-zero section, an index register containing a local index has one of the following formats:

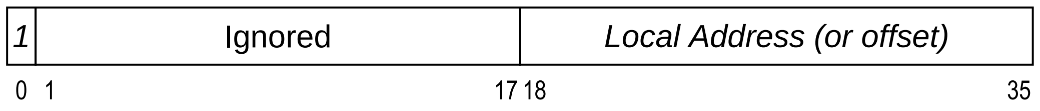


Figure 6-9-3 - Local Index Format (Bit 0 = 1)

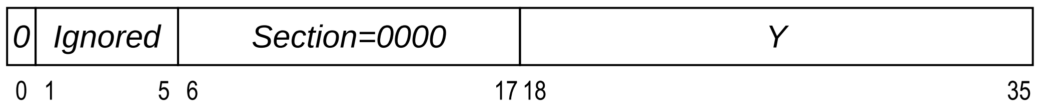


Figure 6-9-4 - Local Index Format (Section 0)

A global index is a 30-bit displacement or address obtained from an index register used in an effective address calculation in a non-zero section, that has bit 0=0 and bits 6-17 non-zero. An index register containing a global index looks like:

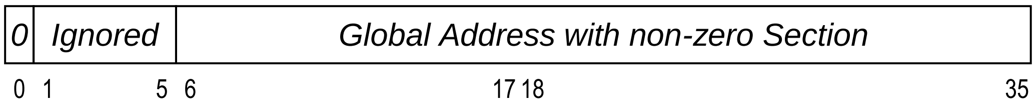


Figure 6-9-5 - Global Index Format

An illegal indirect word is any indirect word in a non-zero section that has both bits 0 and 1 set to a 1. This type of indirect word is reserved for use by future hardware. If an EA-calc encounters this type of indirect word in a non-zero section, it will generate a page fail. The monitor cannot perform any user service as a result of this trap, including trapping to the user, since this would cause possible compatibility problems with future machines. An illegal indirect word looks like:

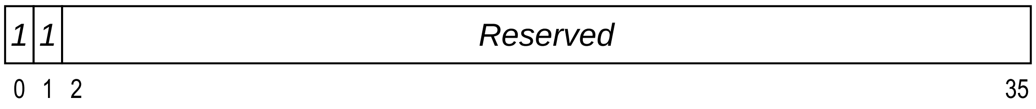


Figure 6-9-6 - Illegal Indirect Word

[i] Clocks for Dophin; Murphy, Dan, DEC Memo.
http://www.bitsavers.org/pdf/dec/pdp10/KXF10_Dolphin/Dolphin_Clocks_for_Dolphin_Mar78.pdf