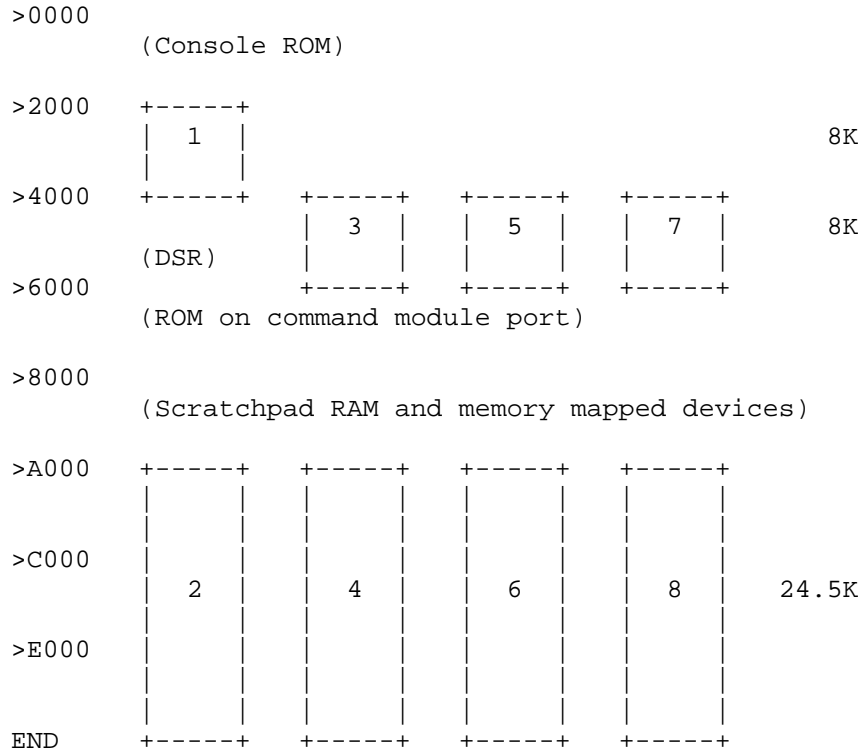


128K EXTENDED MEMORY OPTION
Draft Design

128K of RAM, using the address space for memory expansion and for peripheral devices.



Block 1 is always mapped in.

Block 2 is mapped in on power up. Mapped out by a SBO on CRU addresses >1C00, >1D00, or >1E00. Mapped in by a SBZ on any (all) of those addresses.

Blocks 3 and 4 are mapped in by a SBO to CRU bit >1C00. Mapped out by a SBZ to that address.

Blocks 5 and 6 are mapped in by a SBO to CRU bit >1D00. Mapped out by a SBZ to that address.

Blocks 7 and 8 are mapped in by a SBO to CRU bit >1E00. Mapped out by a SBZ to that address.

Advantages: The CRU-selected blocks at >4000 allow us to load DSR's into RAM with no changes in the system software. Anything other than an >AA in the first byte of those blocks will cause them to be ignored by the DSR search, avoiding lockup when they are used for other purposes.

The block at >2000 is always available and could be used to hold routines for communication between the mapped blocks, and for buffers shared by code in the mapped blocks.

VDP RAM is always available for buffers.

Disadvantages: The mapped blocks cannot refer to each other, e.g., code in block 8 cannot access data in block 3 or branch directly to other code in block 6. (However, some very simple routines could be placed in block 1 to provide such access, with a speed penalty.)

Memory use for a BASIC interpreter:

The BASIC interpreter would be loaded from disk, under the control (at least for the first segment) of a GPL program in a command module.

This module could be specific to the BASIC interpreter, and provide some support routines in GROM and ROM, or could be just a simple boot loader that could be used to load any large program. (Another interpreter or a word processing system, etc.)

A BASIC interpreter comparable to TI Extended BASIC would need to use block 1 for core routines and shared buffers. Blocks 7 and 8 would contain the rest of the interpreter. Blocks 2 through 6 would contain the BASIC program and its data. The line number table needs to be contiguous, and would be kept in block 2. Nearly everything else can be placed wherever there is free memory. Thus the simple routines GETSPACE to get space wherever it is found (looking sequentially in blocks 2 through 6), FREESPACE which deallocates a segment of memory, and GARBAGE, which does garbage collection in each of the blocks in turn, would form the memory management scheme. Statements, symbol table entries, strings, and assembly language segments could all be handled by this memory management system.

Since the statements can be in any block, they must be fetched into block 1 before the interpreter routines in blocks 7 and 8 can access them.

Since the data and symbol tables will be scattered over different mapped blocks, the symbol table search, get-value, and assign-value routines should be in block 1.

Pointers must be two-word values, indicating a CRU base and a memory pointer.

PROBLEMS: main difficulties would be in the design of program LOAD and SAVE and the memory allocation for assembly language subprograms.

BANK SELECTION SCHEME:

Address Range	CRUA >1200	CRUB >1240	CRUC >1280	Selected Bank	A A A A D C B A
>2000 - >3FFF	X	X	X	0	0 0 0 0
>A000 - >BFFF	0	0	0	1	0 0 0 1
>C000 - >DFFF	0	0	0	2	0 0 1 0
>E000 - >FFFF	0	0	0	3	0 0 1 1
>4000 - >5FFF	1	0	0	4	0 1 0 0
>A000 - >BFFF	1	0	0	5	0 1 0 1
>C000 - >DFFF	1	0	0	6	0 1 1 0
>E000 - >FFFF	1	0	0	7	0 1 1 1
>4000 - >5FFF	0	1	0	8	1 0 0 0
>A000 - >BFFF	0	1	0	9	1 0 0 1
>C000 - >DFFF	0	1	0	A	1 0 1 0
>E000 - >FFFF	0	1	0	B	1 0 1 1
>4000 - >5FFF	0	0	1	C	1 1 0 0
>A000 - >BFFF	0	0	1	D	1 1 0 1
>C000 - >DFFF	0	0	1	E	1 1 1 0
>E000 - >FFFF	0	0	1	F	1 1 1 1

PAL Coding:

PAL 14L4

			+-----+	
V1	1			11 A07
/MEMEN	2			12 /CRUCLK
A00	3			13 CRUA
A01	4			14 /V2
A02	5			15 /V3
A03	6			16 /V4
A04	7			17 /CRUST
A05	8			18 CRUB
A06	9			19 CRUC
Power GND	10			20 Power VCC
			+-----+	

$V2 = V1 * MEMEN / A0 * A1 * A2$
 $+ V1 * MEMEN * A1 / A2 * CRUA * CRUB * CRUC$
 $+ V1 * MEMEN * A1 / A2 * CRUA * CRUB / CRUC$
 $+ V1 * MEMEN * A1 / A2 * CRUA * CRUB * CRUC$

$V3 = V1 * MEMEN * A0 * A2 * CRUB * CRUC$
 $+ V1 * MEMEN * A0 * A2 * CRUA * CRUC$
 $+ V1 * MEMEN * A0 * A2 * CRUA * CRUB$

$V3 = V1 * MEMEN * A0 * A1 * CRUA * CRUB$

$CRUST = V1 * MEMEN / A0 * A1 * A2 * A3 * A4 * A5 * A6 * A7 * CRUCLK$

PAL 12L6

		+-----+	
A0S	1		11 /MEMEN
A1S	2		12 PCBEN
A2S	3		13 AC
CRUA	4		14 AB
CRUB	5		15 AA
CRUC	6		16 /RAS1*
AD.A	7		17 /RAS0*
SMQ2	8		18 AD
REFRAS	9		19 Logic GND
Power GND	10		20 Power VCC
		+-----+	

/AD = MEMEN*PCBEN*/A0*/A1
+ MEMEN*PCBEN*/CRUB*/CRUC

/AC = MEMEN*PCBEN*/A0*/A1
+ MEMEN*PCBEN*/CRUA*/CRUC

/AB = MEMEN*PCBEN*/A0*
+ MEMEN*PCBEN*/A1

/AA = MEMEN*PCBEN*/A0*
+ MEMEN*PCBEN*/A2

RAS0* = /AD.A*SMQ2+REFRAS

RAS1* = AD.A*SMQ2+REFRAS