UberGROM 512K ROM/120K GROM Cartridge Boards

This cartridge board significantly expands the capabilities of the TI cartridge port by providing up to 512K of ROM memory using a non-inverted 8K bank-switching scheme. This is the opposite of the scheme originally found on the 16K and 64K cartridge boards built by Jon Guidry, as that method could not extend the banking beyond the 128K maximum imposed by the 74LS379 design. The Guidry boards using the 74LS379 do not impose an expected initialization state on the latch outputs, thus requiring header files in each bank to ensure cartridge initialization. The 512K of bank-switched ROM on the on the new board uses the 74LS378 and does not require the bank inversion used in prior versions of Jon's cartridge board, but it still requires the header files in each bank to ensure stable startup. Finally, the board includes a socket for an ATMega 1284 AVR. Tursi provided code to allow the AVR to be used as 120K of GROM memory space, using up to 16 GROM bases. The combination of these feature changes now allows the board to be used to replicate older ROM/GROM cartridges using 6K or 8K GROMs, with the exception of MBX cartridges, as those would require some additional logic to add 1K of RAM to the board. It is now possible to play some of the unreleased TI titles from a cartridge, as they were originally designed to be used, instead of from disk.

Board Features: Jumpers and Options

The UberGROM boards incorporate a large number of jumpers and option connections intended for customization and/or future expansion. Several of the jumpers are mandatory for normal board operation, and should only be changed in conjunction with potential board customization. Users of ready-made cartridges should never need to change these.

JP1: This jumper connects the Write Enable (WE) signal to U2. For normal board operation, *this jumper must be connected between pins 1 and 2*. Connecting pins 1 and 3 should only be done if attempting to reprogram the flash chip in circuit—a feature possibly requiring board modification.

JP2: Not present. This jumper was eliminated in the final board revisions.

JP3: This jumper connects the Output Enable (OE*) signal to U1 and U2. For normal ROM operation, *this jumper must be connected between pins 1 and 2*. Connecting pins 1 and 3 should only be done if attempting to reprogram the flash chip in circuit.

JP4: This jumper provides Write Protection (WP) for the flash memory in U3 (the AVR). When open, the flash on the AVR may be read and written using the interface documented elsewhere in this document. When closed, the AVR flash is write protected and may not be modified. *This jumper should be closed* when providing final version cartridges to end users.

JP5: This jumper provides the cartridge with a tie point for serial I/O applications. It is not used in the standard software configuration of the AVR, but is provided for use by advanced AVR programmers. The jumper provides pins for Transmit Data, Receive Data, and Ground. *The pins of this jumper should always be open* for normal operation in a cartridge not programmed with a serial I/O function.

JP6: This jumper provides advanced AVR programmers with a connection point for in-circuit AVR programming. The eight pads provide for +5V, two Grounds, MISO, MOSI, RSET*, SS* and SCK between the AVR and the programming device. *There will be nothing attached to these pads* for normal operation in a cartridge unless it is intended for development use.

JP7: This set of jumper pads provides four Analog/Digital Conversion (ADC) lines with an additional ground connection. It is not needed for normal operation but is provided to allow advanced AVR programmers to add ADC capability to their cartridges. When used, they are set to ADC 0-ADC 3. *There will be nothing attached to these pads* for normal operation in a cartridge unless it is programmed for ADC use.

JP8: This jumper controls access to the cartridge port RESET line. *This jumper should always be closed* for normal operation. If the user wants to add an external RESET switch, the jumper should be opened and wires soldered between the switch and the additional pair of pads provided for that purpose. This option provides support for applications needing active external control of the RESET line.

AVR Pad Field: These pads are located under the AVR and provide solder points to connect to the six designated GPIO pins on the AVR. The pads are provided to allow the GPIO to be easily extended outside the cartridge case by advanced users. The pins have the following functions:

Pin 12: GPIO-0
Pin 13: GPIO-1
Pin 20: GPIO-2
Pin 21: GPIO-3
Pin 26: GPIO-4
Pin 27: GPIO-5

Board Features: Required Parts

In addition to the circuit card and the cartridge case, the parts required to assemble the ROM/GROM cartridge board are as follows:

Quantity	Part
2	3-pin jumper post
1	3-pin jumper post (optional)
1	2-pin jumper post, 90-degree bend
1	2-pin jumper post (optional)
1	5-pin jumper post (optional)
1	8-pin jumper post (optional)
1	16-pin DIP socket
1	40-pin DIP socket
1	32 Pin PLCC socket (through-hole)
1	1nF Capacitor

4	.1uF capacitor
1	2200 Ohm resistor
1	68 Ohm resistor
1	ATMega1284 AVR
1	AT49F040 or AT27F040 512K x 8 PLCC Flash ROM (smaller sizes will also work)
1	SN74LS378 or AM25LS07

Board Features: Flash Chip Programming

Some folks have had better luck with the 29F040 as opposed to the 49F040, especially if they are using one of the less expensive adapters to program them. The 49F040 chips program beautifully when using the Yamaichi clamshell type sockets though, and selecting the 32-pin DIP version of the chip as opposed to the PLCC version on the chip selection menu on the programmer may also make a difference. The Advin Pilot series of programmers work very well using this configuration and a clamshell adapter in the DIP programming socket (or with a UPA-84 Universal PLCC adapter. Once properly programmed both chips work in the UberGROM without issues. One other item to note here—smaller 32-pin PLCC flash chips (49F020, 49F010, 49F512, 29F020, 29F010, or 29F512) may also be used when 512K of ROM is not needed for an application. See the cautions under powerup behavior for the 74LS378 and 25LS07 chips when doing so, as chips having a startup bank at other than >6000 will not work with the smaller flash chips.

Board Features: ROM Headers and Bank Switching

The ROM feature in this board is similar to the 16K, 64K, and 128K Bank Switched Cartridge boards designed by Jon Guidry, which, in turn, were extensions of the 16K Bank switching scheme on cartridge boards used by Atari and DataBioTics. This board is different, in that it does not invert the banks.

Board Features: Bank Switching Design

The 74LS378 is tied to TI Address lines A09-A14. This means that it's paying attention to these six bits (when properly enabled and clocked); these can literally be anything from >6000 (0110 0000 0**000 000**0) to >607E (0110 0000 0**111 111**0). Note that the six bits that the 74LS378 controls can be controlled by any write to those particular six bits; therefore we must do two things to "enable" the 74LS378 to perform bank switching:

1) A program writes to a specific address in the cartridge address space (>6000->7FFF) to select an 8K bank. Side note: This means that you could even theoretically write to >7FFE (0111 1111 1**111 111**0), which is the same as writing to >607E (0100 0000 0**111 111**0). However, best practice is to write starting at >6000 up to the maximum capacity of the EPROM (>607E for 512K). See the chart later in this document for valid bank numbers.

2) If the program writes to the cartridge address space (>6000->7FFF) per the restrictions in the preceding paragraph, the console brings console cartridge port signal ROMG* low. This enables the 74LS378 latch (E*), meaning the 74LS378 will respond to the TI console's clock now.

3) The TI console now clocks cartridge port signal (WE*) low, and presents the six bits from the A09-A14 TI address lines to the 74LS378's D0-D5 inputs.

4) **74LS378 (Non-Inverted Bank Switching):** The 74LS378 then presents outputs to its Q0-Q5 outputs. This causes output enabling/disabling EPROM address lines A13-A18 (for a 512K chip), turning on and off specific 8K boundaries in the EPROM and permitting only one of them to be presented to the console. (EPROM address lines A0-A12 = 8K address space, and we only bank A13 and above, which means we're banking with 8K segments.)

The 74LS378 bank-switching scheme is also used on the 512K Red cartridge boards. By using a 74LS377 and a Yellow cartridge board, we gain two additional bits of bank resolution to extend the non-inverted 74LS378 bank-switching scheme to 2048K.

Board Features: Power up Behavior

By default, the 74LS378 should power up in the lowest bank. Since we are using a non-inverted EPROM, this means that bank 0 (the power up bank) is typically the lowest 8K-bank in the chip.

The 74LS378 is a bit unpredictable here, as it does not have a defined power up bank. A 74LS378 should behave as stated above, but if you have issues, try another power-up bank (i.e., if you are using the lowest 8K EPROM segment, try the highest 8K EPROM segment). Chips from the same manufacturer and production lot often react similarly, so a test with one is usually good to determine behavior for all 74LS378s from that manufacturer/lot. Almost all of the 74LS378s I've tested so far (about 300 of them) will start in either the first or the last bank.

The 25LS07 is a bit less predictable here. Each chip will reliably start in a specific bank, but those banks are all over the map. To combat this problem, Tursi built us an excellent bank testing utility that verifies the functionality of every bank of the board, displaying an inline asterisk with the check results for the starting bank. The routine also makes an excellent functionality test for newly-assembled boards, as it quickly identifies any connectivity issues.

The TI-99/4A should scan the power up bank for any ROM cartridge headers present in the EPROM when showing the cartridge selection screen (unless it is a V2.2 console, which requires cartridge-port GROM to also be present).

Board Features: Using the Navarone Cartridge Expander

The Navarone Cartridge Expander is a nice tool, but the method it uses to switch between cartridges is not perfect. Two signal lines are critical to cartridge selection, but it only switches one of them. This has consequences when using an UberGROM, as it doesn't prevent the UberGROM from responding when the switch is set to a different cartridge slot. When using a Navarone cartridge expander, the only cartridge that can be installed is the UberGROM in order to avoid conflicts with the other cartridge expander slots.

When using Red 512K or Yellow 2048K cartridges with the Navarone Cartridge Expander, only two slots should have these types of cartridge inserted (the third slot may contain any regular TI or third-party cartridge. This is a quirk of the bank-switched EPROM boards.

Technical Background: Bank Switching Methods used with the TI

ROM bank switching in the TI world has commonly been accomplished via one of two methods, with minor variations:

1) Writing to the ROM Address space (>6000->7FFF) [Almost all Atari 16K cartridges and DataBioTics 16K cartridges, as well as the Guidry 16/64/128K Cartridge boards]. Bank switched TI cartridges used the same switching method, but on 4K boundaries instead of 8K.

2) Setting a CRU bit and using that to bank switch (DataBioTics 32K and 64K Cartridges).

This board employs **method #1** of banking switching, which in turn, can be done one of two ways:

Non-Inverted method [Atari (and TI) cartridges, as well as 512K and 2048K cartridge boards]:

Non-inverted cartridges can be bank switched by writing to >6000 on upwards, depending on the bank. Note that the value of A15 is effectively insignificant to the TMS9900 due to its use of 16-bit word addressing, so it does not matter what A15 is for our purposes.

The nice thing about a non-inverted bank is that as your EPROM gets larger, you simply keep counting up addresses to write to. I.e., using the chart below, if we had a 2MB EPROM (which is only possible with the 2048K board), its 8K segments would be addressed by writing to the even-numbered addresses between >6000 and >6100.

Non-Inverted Banking Reference for ROM chips from 16K to 512K

```
Binary Address        Hex Addr   8K Bank #  EPROM type (size in K)
--------------        --------   ---------  ----------------------
<- A0 .....  A15 ->
0110 0000 0000 0000 = >6000      (bank 0)   16/32/64/128/256/512K
0110 0000 0000 0010 = >6002      (bank 1)   16/32/64/128/256/512K
0110 0000 0000 0100 = >6004      (bank 2)   32/64/128/256/512K
0110 0000 0000 0110 = >6006      (bank 3)   32/64/128/256/512K
0110 0000 0000 1000 = >6008      (bank 4)   64/128/256/512K
0110 0000 0000 1010 = >600A      (bank 5)   64/128/256/512K
0110 0000 0000 1100 = >600C      (bank 6)   64/128/256/512K
0110 0000 0000 1110 = >600E      (bank 7)   64/128/256/512K
0110 0000 0001 0000 = >6010      (bank 8)   128/256/512K
0110 0000 0001 0010 = >6012      (bank 9)   128/256/512K
0110 0000 0001 0100 = >6014      (bank 10)  128/256/512K
0110 0000 0001 0110 = >6016      (bank 11)  128/256/512K
0110 0000 0001 1000 = >6018      (bank 12)  128/256/512K
0110 0000 0001 1010 = >601A      (bank 13)  128/256/512K
```

0110 0000 0**001 110**0 = >601C (bank 14) 128/256/512K
0110 0000 0**001 111**0 = >601E (bank 15) 128/256/512K
0110 0000 0**010 000**0 = >6020 (bank 16) 256/512K
0110 0000 0**010 001**0 = >6022 (bank 17) 256/512K
0110 0000 0**010 010**0 = >6024 (bank 18) 256/512K
0110 0000 0**010 011**0 = >6026 (bank 19) 256/512K
0110 0000 0**010 100**0 = >6028 (bank 20) 256/512K
0110 0000 0**010 101**0 = >602A (bank 21) 256/512K
0110 0000 0**010 110**0 = >602C (bank 22) 256/512K
0110 0000 0**010 111**0 = >602E (bank 23) 256/512K
0110 0000 0**011 000**0 = >6030 (bank 24) 256/512K
0110 0000 0**011 001**0 = >6032 (bank 25) 256/512K
0110 0000 0**011 010**0 = >6034 (bank 26) 256/512K
0110 0000 0**011 011**0 = >6036 (bank 27) 256/512K
0110 0000 0**011 100**0 = >6038 (bank 28) 256/512K
0110 0000 0**011 101**0 = >603A (bank 29) 256/512K
0110 0000 0**011 110**0 = >603C (bank 30) 256/512K
0110 0000 0**011 111**0 = >603E (bank 31) 256/512K
0110 0000 0**100 000**0 = >6040 (bank 32) 512K
0110 0000 0**100 001**0 = >6042 (bank 33) 512K
0110 0000 0**100 010**0 = >6044 (bank 34) 512K
0110 0000 0**100 011**0 = >6046 (bank 35) 512K
0110 0000 0**100 100**0 = >6048 (bank 36) 512K
0110 0000 0**100 101**0 = >604A (bank 37) 512K
0110 0000 0**100 110**0 = >604C (bank 38) 512K
0110 0000 0**100 111**0 = >604E (bank 39) 512K
0110 0000 0**101 000**0 = >6050 (bank 40) 512K
0110 0000 0**101 001**0 = >6052 (bank 41) 512K
0110 0000 0**101 010**0 = >6054 (bank 42) 512K
0110 0000 0**101 011**0 = >6056 (bank 43) 512K
0110 0000 0**101 100**0 = >6058 (bank 44) 512K
0110 0000 0**101 101**0 = >605A (bank 45) 512K
0110 0000 0**101 110**0 = >605C (bank 46) 512K
0110 0000 0**101 111**0 = >605E (bank 47) 512K
0110 0000 0**110 000**0 = >6060 (bank 48) 512K
0110 0000 0**110 001**0 = >6062 (bank 49) 512K
0110 0000 0**110 010**0 = >6064 (bank 50) 512K
0110 0000 0**110 011**0 = >6066 (bank 51) 512K
0110 0000 0**110 100**0 = >6068 (bank 52) 512K
0110 0000 0**110 101**0 = >606A (bank 53) 512K
0110 0000 0**110 110**0 = >606C (bank 54) 512K
0110 0000 0**110 111**0 = >606E (bank 55) 512K
0110 0000 0**111 000**0 = >6070 (bank 56) 512K
0110 0000 0**111 001**0 = >6072 (bank 57) 512K
0110 0000 0**111 010**0 = >6074 (bank 58) 512K
0110 0000 0**111 011**0 = >6076 (bank 59) 512K
0100 0000 0**111 100**0 = >6078 (bank 60) 512K
0100 0000 0**111 101**0 = >607A (bank 61) 512K
0100 0000 0**111 110**0 = >607C (bank 62) 512K

```
0100 0000 0111 1110 = >607E      (bank 63)  512K
```

<u>Inverted method [DataBioTics cartridges, as well as the Guidry 16/64/128K Cartridge Boards):</u>

Inverted cartridges can also be bank switched by writing from >6000 upwards, just like the previous example. However, the process is a bit different. As above, note that the value of A15 is effectively insignificant to the TMS9900 due to its use of 16-bit word addressing, so it does not matter what A15 is for our purposes.

To figure out an inverted bank switch scheme, you cannot do as you can with a non-inverted scheme and start counting upwards for the addresses. You must first know what size EPROM you have. I will give a 64K board as an example.

Non-Inverted Banking Reference for a 64K ROM chip:

```
Binary Address        Hex Addr   8K Bank #  EPROM type (size in K)
--------------        --------   ---------  ----------------------
<- A0 .....  A15 ->
0110 0000 0000 0000 = >6000      (bank 0)   16/32/64
0110 0000 0000 0010 = >6002      (bank 1)   16/32/64
0110 0000 0000 0100 = >6004      (bank 2)   32/64
0110 0000 0000 0110 = >6006      (bank 3)   32/64
0110 0000 0000 1000 = >6008      (bank 4)   64
0110 0000 0000 1010 = >600A      (bank 5)   64
0110 0000 0000 1100 = >600C      (bank 6)   64
0110 0000 0000 1110 = >600E      (bank 7)   64
```

Inverted Banking Reference for a 64K ROM chip:

```
Binary Address        Hex Addr   8K Bank #  EPROM type (size in K)
--------------        --------   ---------  ----------------------
<- A0 .....  A15 ->
0110 0000 0000 0000 = >6000      (bank 7)   16/32/64
0110 0000 0000 0010 = >6002      (bank 6)   16/32/64
0110 0000 0000 0100 = >6004      (bank 5)   32/64
0110 0000 0000 0110 = >6006      (bank 4)   32/64
0110 0000 0000 1000 = >6008      (bank 3)   64
0110 0000 0000 1010 = >600A      (bank 2)   64
0110 0000 0000 1100 = >600C      (bank 1)   64
0110 0000 0000 1110 = >600E      (bank 0)   64
```

Note that the banks are simply reversed. Let's look at bank 0 a little closer:

For bank 0, the non-inverted 64K "bank 0" uses a "000 write" on the A12-A14 bits in the cartridge space (>6000->7FFF). It is presented to the 74LS379 and not inverted in the output by the 74LS379. Therefore, "000" in = "000" out. By default, this will present the bottom 8K of the 64K chip as "bank 0" using a non-inverted method.

For bank 1, the inverted 64K “bank 0” also uses a “000 write” on the A12-A14 bits in the cartridge space. The bits on A12-A14 presented to the 74LS379’s inputs are promptly inverted on the 74LS379’s outputs. Therefore, “000” in = “111” out. So, that now means that our “bank 0” is now officially the last 8K segment in the 64K ROM.

Bottom line: The inverted bank switching method simply “flips” the order of the 8K banks to where the first 8K bank is actually the highest bank on the EPROM, and the last 8K bank is actually the lowest bank on the EPROM.

Inverted bank switched EPROMs can be converted to non-inverted EPROMs simply by flipping the 8K segments. In an inverted 64K EPROM, the order of the 8K segments is:

- 7 6 5 4 3 2 1 0

To make a program in this bank switched EPROM work in a non-inverted EPROM, flip the 8K segments to:

- 0 1 2 3 4 5 6 7

Technical Background: Sample ROM Cartridge Header

Here is a simple cartridge header used to make your own ROM cartridge. Note that this example does not include bank switching:

```
********************************
* ROM CARTRIDGE HEADER         *
********************************

       DEF SFIRST,SLAST,SLOAD
       UNL

* ROM HEADER
       AORG >6000    * START ADDRESS OF CARTRIDGE ROM

SFIRST EQU $
SLOAD  EQU $

GRMHDR BYTE >AA      * INDICATES A STANDARD HEADER
       BYTE >01      * VERSION NUMBER
       BYTE >01      * NUMBER OF PROGRAMS (OPTIONAL)
       BYTE >00      * NOT USED
       DATA >0000    * POINTER TO POWER-UP LIST (CAN'T USE IN
CARTRIDGE ROM)
       DATA PROG     * POINTER TO PROGRAM LIST
       DATA >0000    * POINTER TO DSR LIST
       DATA >0000    * POINTER TO SUBPROGRAM LIST
       DATA >0000
```

```
        DATA >0000
PROG    DATA >0000     * NO NEXT MENU ITEM
        DATA MAIN      * PROGRAM START ADDRESS FOR THIS MENU ITEM
        BYTE 12        * LENGTH OF TEXT FOR MENU SCREEN
        TEXT 'PROGRAM NAME'
        EVEN

MAIN    EQU $
* YOUR CODE GOES HERE

FINISH EQU $
SLAST   END
```

Board Features: UberGROM Interface

The UberGROM Interface is a GROM-compatible device built on the Atmel Mega-AVR series of microcontroller. It is self-contained and can attach to the TI-99/4A GROM bus with no additional hardware (although filter capacitors are recommended on the power supply lines). Although the code can be adapted to any size of AVR, the current build is shipped with the Atmel ATMEGA128 for its large flash memory size and rich peripheral interface.

The TI-99/4A has three types of memory built-in: CPU Memory, which is directly addressable by the CPU, VDP memory, only addressable through the VDP, and GROM memory, only addressable through the GROM memory base. Most homebrew cartridges are created using CPU memory, due to the difficulty in reproducing TI's proprietary GROM interface, and the inaccessibility of VDP memory from outside the system. This chip is intended to resolve this difficulty and allow GROM to be added easily to any cartridge.

The TI system provides direct connection for 64k of GROM space, divided into blocks of 8k each (most official GROMs used only 6k of this 8k space). The first three blocks are allocated to the operating system (GROM 0) and TI BASIC (GROMs 1 and 2). The remaining 48k of space is intended for use by cartridges.

The system also provides the concept of paging to the GROM space. Rather than an explicit paging action being required, the CPU simply interfaces through a different set of addresses (a different GROM 'base'). The console scans 16 of these bases when building the selection list, as the cartridge port only supports the first 16 GROM bases, but the memory map includes space for up to 256 bases. At 64k each, this represents a potential memory space of 16MB!

There are a few caveats to this, however. The three console GROMs are wired so that they always respond to any base, thus, only 48k is available in each base available to the cartridge port. In addition, there's no single microcontroller yet available that can provide this much memory, so multiple chips and additional circuitry is required to go to this entire range.

A second caveat is speed. While a ROM access to the cartridge base happens in 4 cycles (and is that slow only due to the multiplexer), a GROM access can delay responses for as much as 30 times longer. This delay is unrelated to the performance of the microcontroller, as all console GROMs add their time to the delay, even if they are not required to respond to the access.

This microcontroller attempts to provide a medium between flexibility of these spaces, memory usage, and peripheral interface. It provides re-writable GROM memory, GRAM, EEPROM, GPIO, Analog-to-Digital converters, and a TTL serial port (USART).

Board Features: UberGROM Specifications

Core: Atmel AVR ATMEGA1284 8-bit RISC @ 8MHz (internal clock)
GROM: 120k (in 8k banks, supports 16 access bases)
RAM: 15k (8k and 7k bank)
EEPROM: 4k
GPIO: 4 pins
ADC: 4
UART: 1
EEPROM Write Time: 3.4ms
EEPROM Erase Endurance: 100,000 cycles
Flash Erase Time: 72ms (entire 8k block)
Flash Write Time: 4.5ms (256 byte page)
Flash Erase Endurance: 10,000 cycles
Clock accuracy: +/- 2.5% @ 25C
Current Draw: approx. 8.5mA @ 25C

See the ATMega1284P Datasheet for more specifics.

The system can be used as a simple GROM device, or it can be used as a peripheral interface to external devices. Full source is available so that you can adapt it to your own usage. It's limited only by your own design!

This device is designed to work on the concept of memory-mapping. For each of the available GROM slots in the TI system, a configuration register specifies whether memory or peripheral is accessed at that address. All access to the device is done via GROM.

Technical Background: Terminology

Graphics Read Only Memory (GROM) – TI's proprietary parallel, auto-incrementing memory device. Alternately, it is the memory range that would contain a single GROM on the TI. This device emulates many GROMs in one unit.

Base – The group of addresses that selects a bank of GROMs. Base 0 is at >9800 for 'read data', with offsets of 2 for 'read address', 1024 for 'write data', and 1026 for 'write address'. Subsequent bases are offset by 4 from the first, so, Base 1 is at >9804, Base 2 is at >9808, etc. This device supports 16 bases.

Slot – A term specific to this device to indicate the programmable memory address into which any peripheral may be configured.

Technical Background: The TI-99/4A GROM Interface

GROM is a parallel, auto-incrementing memory device. No matter how many GROMs are physically attached to a base, all GROMs see all GROM accesses and all of them respond in some manner. This section covers only the software interface. For information on the hardware interface, it is recommended to consult additional sources, such as Thierry Nouspikel's excellent reference at http://www.nouspikel.com/ti99/titechpages.htm.

Four addresses are dedicated in the 9900's address space for communicating with GROM. This is called the GROM 'base'.

>9800 – read data
>9802 – read address
>9C00 – write data
>9C02 – write address

Normal operation of the GROM starts by writing an address. This is done by sending the two bytes of the desired address to the write address base (with simple MOVB opcodes), sending the least significant byte first. Once the address has been sent, the data can be read by reading from the Read Data base (again, with MOVB). The GROM automatically increments its internal address every time a byte is read, so that subsequent bytes can be read without having to set the address again. When reading, only the GROM that actually contains the data will present it, but all GROMs participate in the cycle.

The auto-increment of the GROM address has one caveat, when it comes to thinking of the address space. Although there are 16 bits of address space, each GROM tracks only 13 bits of address internally (for 8k of space). The other 3 bits are latched as a 'device select'. This means that when an address reaches that 8k boundary, it wraps around to the beginning of that GROM again, rather than going into the next boundary. This is a bit counter intuitive, but it makes sense when you remember that is it normally a bank of 8k (or 6k!) chips all responding to the same bus. This internal address management allows the chips to be used without any external hardware for chip selection, but makes it difficult to automatically count from one chip to the next. The only way to change which 8k block is active is to write a new GROM address. It is important to be aware of this behavior with the peripherals, which may wrap around back to GROM addresses if your code does not follow the described rules.

There is an advanced configuration option to disable this behavior and allow the address to increment within the space of the device (>6000->FFFF).

GROMs are notoriously slow devices, but unlike the VDP, the software need not be aware of delays when accessing them, as they will block the CPU for as long as is needed to complete the operation. Although this microcontroller is much faster than the real GROMs for most operations (flash writing excluded), the delay still extends to the slowest GROM in the system.

Read Address is used to read the current address in the GROMs back, and again, the LSB is sent first. Reading the address corrupts it, and so a new address must be written before you may read again. Due to the GROM internally prefetching data, the retrieved address will be one higher than expected. Note that reading address on peripheral blocks may not return the expected address. This should only be used on banks emulating GROMs or GRAMs (or real GROMs, of course).

Note that this device does not return addresses; therefore, there must always be at least one real GROM in the system.

Write Data is not used on normal GROMs, but may be used with GRAM and with this microcontroller in the same manner. The effect of prefetch on the address may be safely ignored when writing, but as the standard operation for this was never defined by TI, it is not recommended to mix reads and writes without resetting the address in-between.

To access a different GROM base, just add 4 to the desired address for each base. For instance, the above addresses access base 0, base 1 would use >9804 for Read data, base 2 would use >9808, and so forth. Remember that the console GROMs (>0000 - >5FFF) respond regardless of the base number.

A note on device interfacing

The TI console includes limited support for multiple GROM bases, and a 'Review Module Library' option. However, it determines whether to activate this by comparing the first 31 bytes of cartridge space, starting at >6000. There is a bug in this loop, in that if multiple bases ARE detected, but no valid cartridge header is detected, the console will go into an infinite loop while building the module list. In order to make it less likely to cause this behavior by accident, all peripheral spaces on this device start at a >0020 offset, and the first 32 bytes. Thus, only GROM, RAM or EEPROM can actually be at >6000.

If you want to avoid the console detecting multiple GROM bases, and thus prevent "REVIEW MODULE LIBRARY" from appearing on the cartridge menu, simply select the same memory at GROM address >6000 in both bases 9800 and 9804 – this is all that the console checks. This costs you one slot, but there are plenty of spare slots even if you wish to map all peripherals.

UberGROM Features: EEPROM/Configuration space

EEPROM stands for 'electrically erasable programmable read-only memory'. The EEPROM in this device is a non-volatile memory that can be read and written a single byte at a time, using standard GROM accesses, although writes may be notably slower. The first few bytes of the EEPROM contain paging and configuration registers, the rest can be used by your own application. This data is non-volatile and does

not require power or a battery to retain. (Although, as EEPROM data can fail over repeated use, it is recommended that a checksum or other system always be used to validate the stored data.)

It should be noted that the EEPROM is rated for a limited number of writes to a particular byte, specifically 100,000. This is adequate for tasks such as saving configuration data, high scores for games, , occasional changing of the cartridge by a user, etc., but not for continuous access tasks, such as bank switching during execution. As such, the configuration block here is intended to be set once by the cartridge creator, or by a user loading a new cartridge into their device, but not continuously during runtime. (Bank-switching with a cost of 3.4ms per switch would be unreasonably slow anyway). Rather than switching devices in and out, simply use the multiple GROM base feature to map all devices into unique addresses, and they will all be available to your application simultaneously.

The configuration EEPROM is available in top 2k of the last GROM base (even if GROM bases are not being used), starting at GROM address >F800, regardless of configuration. (Base 15 is accessed starting with CPU address >983C). This mapping is fixed and cannot be moved, as it is the only way to access the banking configuration. This means that GROM space in base 15 is limited to 38K. (However, if banking is not used, the entire 40k space of bank 0 is still available). As the EEPROM is larger than 2k (see specifications), then it must be mapped into another base to access the rest of the data.

The configuration EEPROM defines the following registers, which start at >F800 in base 15 in GROM space (always), or at >0000 in the GROM base that EEPROM is mapped to.

0000 – Configuration flags. The following bits are defined. Unused bits should be set to 0.

> 01 – Enable GROM bases when set. When set to disabled, the GROM will respond to all bases, the same way that the console GROMs do. In this mode, only 16 bytes are used for mapping. When set, the GROM will respond to the first 16 GROM bases independently, and 256 bytes are used for mapping. Note that regardless of this setting, the fixed EEPROM space is available on base 15 at >F800
>
> 02 – Disable Recovery program when set. On first powerup, the chip will use the powerup vector at >E000 to test the keyboard, and if the Space Bar is being held, it will load a simple test GROM at >6000 instead of whatever is configured. This behavior is probably not desired for a production cartridge, but should not affect any other. Set this bit to prevent it. Note that if you set this bit, you will need an external way (such as a ROM or boot menu) to load the loader program to change anything on the cartridge, as the internal loader will not be accessible.
>
> 04 – Allow rollover. TI manufactured GROMs maintain an address counter within an 8k GROM. That is, when they reach the end of their slot, they wrap back to the beginning (so, for instance, >7FFE, >7FFF, >6000, >6001, etc.). Many GROM emulation devices did not have this behavior; they maintained a single 16-bit address counter (thus allowing 0x7FFE, 0x7FFF, 0x8000, 0x8001, etc.). By default, this device emulates the original behavior. If you set this bit, then roll-over into the next slot will be permitted. In order to prevent conflict with the console GROMs, roll-over from 0x5FFF->0x6000, and from 0xFFFF->0x0000 will not occur in either case. Note: Because this

device does not provide address read-back (this functionality is provided still by the console GROMs), if you roll over from one GROM slot to the next the system GROM address will be incorrect if read back. Therefore, ensure that this will not be a problem for your application before using this bit. For instance, it may be a problem for GPL applications which read back frequently, but less so for data copy loops and the like. (Example: If you allow the GROM read to increment from >7FFF to >8000, data will continue to come from the correct place, but reading the GROM address will return >6000 instead of >8000, because the console GROMs do not support this operation.)

0001 – Inverse bits to register 0000. If this value is not the same as a bitwise invert of register 0000, then the value in 0000 is considered invalid and not used. (0x00 is assumed).

0002 – Map configuration for GROM slot >0000. Note: this slot is reserved for the TI operating system, and any override placed here, in any base, is ignored. (For future expansion)

0003 – Map configuration for GROM slot >2000. Note: This slot is reserved for the console TI BASIC, and any override here, in any base, is ignored. (For future expansion)

0004 – Map configuration for GROM slot >4000. Note: This slot is reserved for the console TI BASIC, and any override here, in any base, is ignored. (For future expansion)

0005 – Map configuration for GROM slot >6000.
0006 – Map configuration for GROM slot >8000.
0007 – Map configuration for GROM slot >A000.
0008 – Map configuration for GROM slot >C000.
0009 – Map configuration for GROM slot >E000.

000A – Inverse bits to register 0002. If this value is not the same as a bitwise invert of register 0001, then the value in 0001 is considered invalid and not used.
000B – Inverse bits to register 0003.
000C – Inverse bits to register 0004.
000D – Inverse bits to register 0005.
000E – Inverse bits to register 0006.
000F – Inverse bits to register 0007.
0010 – Inverse bits to register 0008.
0011 – Inverse bits to register 0009.

0012-0101 – Map configuration for GROM slots >0000->E000 in cases where multiple GROM bases are enabled. 16 bytes are used for each slot in the same manner as defined above for the first base. Note that when multiple GROM bases are enabled, the TI console GROM requires at least ONE valid cartridge GROM to be available, otherwise the title selection menu will hang. If multiple GROM bases are not enabled, this memory is free for user data.

0102-max – Free for user data.

The following table lists positions for each combination of GROM base (down), and memory slot (across) to load into the EEPROM. All address offsets are in hexadecimal, and the inverted byte offset is gained by adding ‘8’ to the listed value.

	0	2000	4000	6000	8000	A000	C000	E000
0	2	3	4	5	6	7	8	9
1	12	13	14	15	16	17	18	19
2	22	23	24	25	26	27	28	29
3	32	33	34	35	36	37	38	39
4	42	43	44	45	46	47	48	49
5	52	53	54	55	56	57	58	59
6	62	63	64	65	66	67	68	69
7	72	73	74	75	76	77	78	79
8	82	83	84	85	86	87	88	89
9	92	93	94	95	96	97	98	99
10	A2	A3	A4	A5	A6	A7	A8	A9
11	B2	B3	B4	B5	B6	B7	B8	B9
12	C2	C3	C4	C5	C6	C7	C8	C9
13	D2	D3	D4	D5	D6	D7	D8	D9
14	E2	E3	E4	E5	E6	E7	E8	E9
15	F2	F3	F4	F5	F6	F7	F8	F9

Map Configuration

The map configuration consists of two nibbles. The first nibble indicates the device type to be mapped in, devices are described below. The second nibble indicates which page of the device, if it has multiple pages. For instance, a map of >12 indicates GROM page 2 should be mapped. The number of pages available is the device memory size divided by 8k, rounded up.

Device ID	Type	Access	Pageable (Y/N)
0	RAM	Read/Write	Yes
1	GROM	Read Only	Yes
2	EEPROM	Read/Write	No
3	GPIO	Peripheral	No
4	ADC	Peripheral	Yes
5	UART	Peripheral	No
6	Flash	Peripheral	No
7	Timer	Peripheral	No

Non-pageable devices should set a page of 0 for compatibility.

UberGROM Functions: Device Type 0 – RAM

This microcontroller provides a block of ordinary RAM memory which may be read or written through the standard GROM interface (as if GRAM). No locking or erasing is required, and the data is available upon paging. The data is volatile, however, and will be lost if the console is turned off or the cartridge is removed from it.

It is even possible to load GROM programs to the RAM block and have the interpreter execute them, if desired. However, because this RAM is on the GROM bus, the TI CPU cannot execute 9900 assembly code from it.

Each RAM page is 8k in size – in this controller the second page is only 7k in size.

UberGROM Functions: Device Type 1 – GROM

This is the standard GROM data, and will generally consist of many available pages of read-only memory. This memory is stored in the microcontroller's flash and is non-volatile; it does not require power to retain. It is not trivially re-writable, you must page in the Flash device and explicitly erase and re-write blocks of data. In this mode (type 1), however, it will behave like standard GROM. Writes to the GROM data will be ignored.

Note: There is one limitation on the placement of data in GROM. A Powerup link placed at >E000 in the first GROM base will not execute unless the recovery code is disabled in the configuration space. See the Recovery code section.

UberGROM Functions: Device Type 2 – EEPROM

See the section above about EEPROM/Configuration data for full details. This page can be mapped into a block for convenience or to access more than 2k of EEPROM, if available. Otherwise, it is always available at GROM address >F800 in base 15 (even if GROM bases are not enabled in configuration).

The EEPROM space will be read-only until an unlock sequence is written to it. To unlock it, simply write to >FFFF on base 15 three times, with the following bytes. Note that any other GROM write access to EEPROM space will break this sequence (but by limiting it to EEPROM space it should still work from GPL): Write 0x55, then write 0xAA, then write 0x5A.

You can lock the EEPROM again by writing any other value to >FFFF on base 15. This is recommended after a modification to prevent resets and glitches from accidentally modifying EEPROM data.

Be careful if accessing user EEPROM in the default mapping on base 15, since only offsets 0x0000-0x07FE are available – offset 0x07FF overlaps the lock control address. This does not apply when EEPROM is mapped anywhere else.

UberGROM Functions: Device Type 3 – GPIO

GPIO stands for General Purpose Input/Output. This microcontroller supports 4 GPIO pins. Each pin is a 5-volt pin which may be configured for input or output. When input, it may also be configured to activate an internal pull-up resistor, to reduce external parts count.

The device registers start at >0020 in whichever page they are mapped into. In addition, the access space deliberately repeats over a range to allow repeated access to the I/O pins without needing to change the GROM address between each one. (Even though this is a device access, the GROM address must still auto-increment).

The least significant bit indicates GPIO-0 in all cases.

0020 – Configuration bits for Input or Output. Each bit, starting with the LSB, is set to 0 for input, or 1 for output. On reset, all GPIO are set to input with no pull-up – your software must configure it.

0021-1FFF – GPIO pin access. When reading, it returns the current input state of each pin, with the LSB set to GPIO-0. This is true whether the pin is in input or output mode. When writing a pin set for output, the pin will output 5v if a 1 bit is set or 0v if a 0 bit is set. When writing a pin set for input, the pull-up resistor is enabled if a 1 bit is set, or disabled if a 0 bit is set.

The large range is used to allow multiple accesses without resetting the GROM address, but note that to read or write continuously, you must reset the GROM address before it leaves this range.

UberGROM Functions: Device Type 4 – ADC

ADC stands for Analog to Digital Conversion. The microcontroller supports multiple analog input pins with a sense range from 0 to 5v, with 8-bit resolution (depending on the system configuration.) The ADC 'page' mapped indicates which of the 4 ADCs will be accessed.

The ADCs are mapped starting at an offset of >0020. The entire memory space (>0020 - >1FFF) repeats to allow them to be read continuously without needing to reset the GROM address. The page number specified indicates the ADC that you wish to access.

Reading the ADC address triggers the conversion immediately, which will convert in approximately 104uS. A value of 0 indicates a value near 0v, while 255 indicates a value near 5v.

Note: the first conversion after power up will take approximately 200uS.

Tip: By placing the ADCs in different GROM base bases, but at the same GROM address, you could read all ADCs without having to update the GROM address just by changing which CPU address you access GROM through.

UberGROM Functions: Device Type 5 – UART

UART stands for Universal Asynchronous Receiver/Transmitter, and this is a serial communication device. The UART in this microcontroller can handle high speed bidirectional traffic, and is equipped with two 255 byte buffers, one each for receive and transmit. Despite this, higher speeds may have limited value on the TI due to its ability to actually service the buffers without losing data.

The UART is mapped starting at an offset of >0020. Also, the read and write bases are allocated a block of repeating memory to allow them to be accessed continuously without needing to reset the GROM address.

0020 – Status/Configuration flags:

> 01 – (Read only) When set, there is data available to be read. If the buffer fills and more characters arrive, the additional characters are dropped (and set the Data Overrun bit).
>
> 02 – (Read only) when set, there is room in the output buffer for at least one byte of data.
>
> 04 – (Read Only) TX Buffer Empty – Indicates that the transmit buffer is completely empty (i.e., all data is sent)
>
> 10 – (Read only) Frame error - This bit is set if the USART had a Frame Error. I.E. when the first stop bit of the next character in the receive buffer is zero. There is no way to tell which character in the buffer this applied to, and due to the behavior of the hardware, in some cases it may not clear until a valid character is received. This bit may be of limited value.
>
> 20 – (Read only) Data Overrun – this bit is set if the receive buffer is full and a new character is received. At very high bit rates characters may be lost without moving to the buffer, in which case this bit is not set, but this case should be very rare.
>
> 40 – (Read only) Parity Error – this bit is set if a received character had a parity error when it was received. For the same reasons as Frame Error, this bit may be of limited use.

0021 – Line setting flags:

> 01-02 – Number of bits (11=8, 10=7, 01=6, 00=5).
>
> 04-08 – Parity Generation (00=none, 01=none, 10=even, 11=odd).
>
> 10 – Stop bits (0=1 stop bit, 1=2 stop bits).
>
> 20 – 2X Mode (doubles the baud rate at the expense of less accurate clock tolerance).

0022-0023 – Speed setting. This is a Little-Endian, 12-bit value loaded into the AVR's UBRR register to set the baud rate of the serial device. To calculate this value, use: UBBR = (8,000,000/(16*BAUD))-1. The valid range is from 0 (representing about 500,000bps) to 4095 (representing about 122bps). Rates are

doubled (but clock recovery accuracy halved) if the 2X bit is enabled. You must write these bytes in order – LSB then MSB. The change won't take effect until the MSB is written.

0024 – Number of bytes in the RX buffer. This can be used to optimize reads from the buffer – even if new bytes are added, this much data is guaranteed to be present.

0025 – Number of bytes in the TX buffer. This can be used to optimize writes to the buffer – this much space is guaranteed to be present.

0100-0FFF – Write byte (goes to buffer before transmission). If the TX buffer is full, the byte is silently dropped – you should check the buffer before writing. Note this GROM range is smaller than for reading.

1000-1FFF – Read byte (from buffer if buffering is on). Note that this will clear error bits in the status/configuration register. If no character is available in the buffer, the returned byte is not guaranteed – you should check the buffer before reading.

UberGROM Functions: Device Type 6 – Flash Controller

The flash controller provides write access to the GROM memory space. The GROM slot is the same as for the GROM device type. In this mode, you can erase the page and rewrite data 256 bytes at a time. This mode can be disabled with the external Write Protect hardware pin.

The flash controller is mapped starting at an offset of >0000.

The process is fairly simple:

1) Check the result code for write protect (abort if set)
2) Set the page select register and slot select register
3) Write the erase command to the command register (will delay until finished)
4) Check the result code for errors
5) Write the new page data to the write buffer
6) Write the program command to the command register (will delay until finished)
7) Check the result code for errors
8) Repeat until all pages are written

0000-00FF – Write buffer (one page worth – write only). You MUST write this page sequentially, and always write an even number of bytes, for correct results. This buffer is write-only.

0100 – Page select in GROM slot (0-31)

0101-0102 – The Write command register. Write the command in 0201 and the inverse in 0202 (write only):

31/CE = Erase block – causes the page (256 bytes) to be erased to 0xFF
D2/2D = Write buffer – writes the contents of the write buffer to the current page

0103 – Result Code (read-only)

0 = Idle
1 = Busy
2 = Write Protected
3 = Command Error

0x0104 – GROM slot (0-15)

UberGROM Functions: Device Type 7 – Timer

The timer is a free-running, real-time tick counter that can be used to time events independent of the console operating speed, and with a higher resolution than the VDP interrupt (and with a quicker readback than the 9901 timer). The timer runs at 7812.5Hz (+/- 2.5%), and is 16-bits in size.

The timer is mapped starting at an offset of >0020. Also, the read-back registers are allocated a block of repeating memory to allow them to be accessed continuously without needing to reset the GROM address.

Because the timer is 16-bits in size, two reads are required to read the data. Every even numbered address is the LSB, and every odd numbered address is the MSB. The timer values are updated when the LSB is read, thus, the MSB will always be associated with the LSB. Because of this, you cannot simply read the MSB, the value will never change.

0020-1FFF – Timer value, 16-bit little-endian (LSB, MSB), repeating (read only).

Recovery Program

The device includes a small recovery program to facilitate re-loading the chip even if suitable software is not otherwise installed. A PowerUp link is always inserted on the FIRST powerup only, at the >E000 GROM base. (That is, it will not be re-executed when FCTN-= or even a TI Hardware Reset is executed, the system must be power cycled to re-trigger it). If the user holds the Space Bar during the power-on, the GROM at >6000 on base 0 is replaced with the recovery program, which contains a copy of Easy Bug, and the LOAD PROGRAM IMAGE function from Editor/Assembler. This allows troubleshooting and the ability to load any piece of software into the system for the purpose of configuring the chip.

If this functionality is not desired, for instance, either you need a real power up routine in the GROM at >E000, or you are distributing a cartridge and do not want users to be able to activate this mode, simply set bit value >02 in the configuration bits. Once you do this, it will be impossible to reactivate this recovery program without updating this configuration bit again – since the GROM chip usually occupies the cartridge base, this means you may need an alternate way to load software (such as a ROM beside the GROM, or boot software like the startup MENU program.)

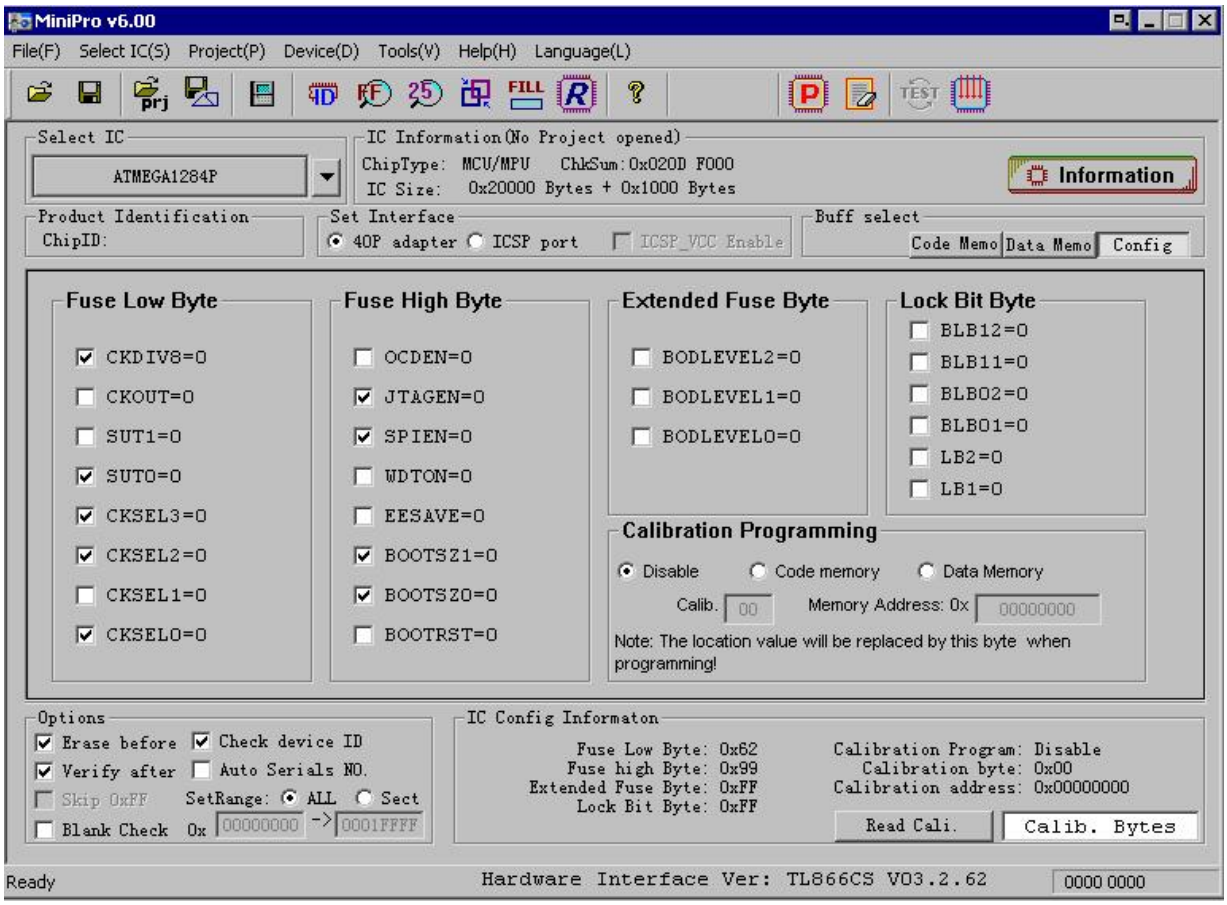
The recovery program contains program software copyright by Texas Instruments, and distributed under license. TI makes no warranty with respect to the programs and is under no obligation to provide any support or assistance with respect to the programs. TI is under no obligation to provide upgrades to the

programs. No liability is accepted on the part of Texas Instruments or the author with respect to use, copying or distribution of the programs. This software has been modified from its original form.

Initial Programming of an ATMEL ATMega1284P

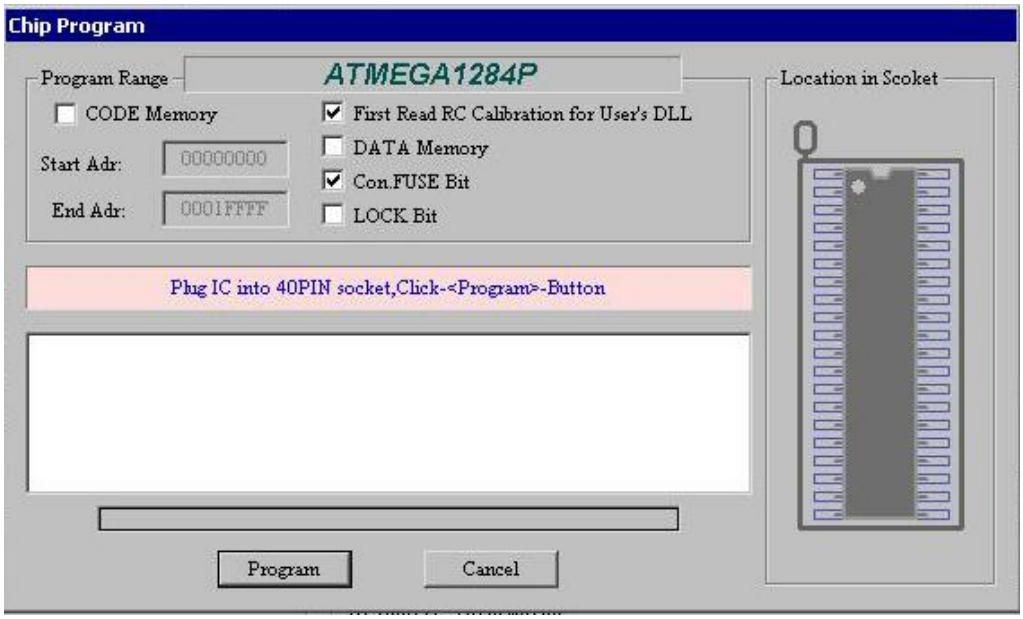
Tony Knerr used the following process to initially program the ATMEL ATMega1284P chips he used. His process requires two Programmers, a MiniPro and an AVRISP Mk II. The steps he followed are provided here as a template for individuals using the same or other programmers when initially preparing AVR chips for the UberGROM board.

Step 1: Use the MiniPro to set the configuration, selecting the 1284P chip and the MiniPro's default configuration options for the chip, shown in the screenshot below:

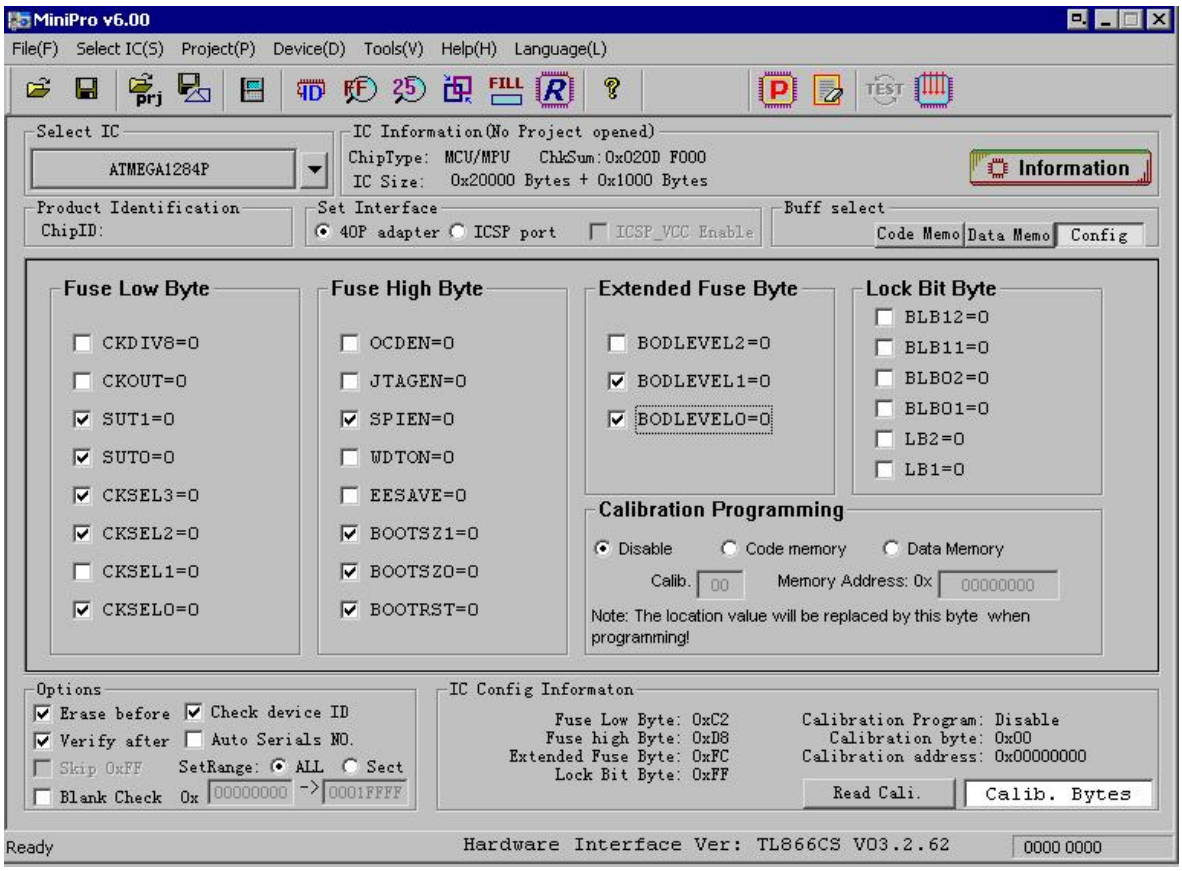


Click "Device" and then "Program" to complete this step.

Step 2: Set the check boxes to match the screenshot below and click "Program"



Step 3: Change the configuration options as shown in the screenshot below and then click "Device" followed by "Program." Then click "Program" again from the Programming screen.



Step 4: Remove the AVR chip from the MiniPro and put it into the AVRISP MK II programmer. Start ATMEL Studio and click on "Tools" and then "Device Programming."

Step 5: Choose AVRISP Mk II under “Tool” and click “Apply.” Click “Read” under “Device Signature.”

Step 6: Click “Interface Settings” and move the “ISP Clock” slider to 8.239KHz and click “Set.”

Step 7: Click “Memories” on the left and choose your Flash and EEPROM files.

Step 8: Click “Program” in the EEPROM section and wait until it completes verification.

Step 9: Click “Program” in the Flash section to start loading the GROM data. You will get multiple timeout errors here; just click “Continue Waiting” until it’s done.

This process takes about 40 minutes, but by following it, you end up with a working chip. Two programmers are required because the AVRISP Mk II doesn’t have the oscillator to provide the external clock necessary to force a full chip reset.

Using just the AVRISP Mk II and ATMEL Studio, the following steps will successfully program a new ATMEL ATMega1284P. Note that this process works only once per chip due to the missing oscillator needed to provide the external clock to reset the chip if something goes wrong on the first attempt. Subsequent attempts with the chip will give you the 0xC0 error until you’ve reset the chip using another means.

Step 1: Load ATMEL Studio and set up Device Programming for “AVRISP Mk II” and targeting the “ATMEL1284P” and “ISP.” Click “Read” under the Device Signature setting.

Step 2: Click “Interface Settings” and move the slider to set the ISP Clock to 64KHz and click “Set.”

Step 3: Click “Fuses” and set the options as follows:

BODLevel: 4V3

OCDEN: Unchecked

JTAGEN: Unchecked

SPIEN: Checked

WDTON: Unchecked

EESAVE: Unchecked

BOOTSZ: 4096W_F000

BOOTRST: Checked

CKDIV8: Unchecked

CKOUT: Unchecked

SUT_CKSEL: INTRCOSC_6CK_0MS

Step 4: Make sure “Auto Read” and “Verify After Programming” are checked and click “Program.” It should program OK.

Step 5: Click “Read” to verify that the settings you have just made show up. Note: failure to execute this step will cause all subsequent steps to fail.

Step 6: Click “Memories” and fill in your file for “EEPROM (4KB)” (your chip’s “Data Area”). Check “Verify EEPROM after Programming” and click “Program” in the “EEPROM (4KB)” section. It should program OK.

Step 7: Fill in your file for “Flash (128KB),” uncheck “Erase Device before Programming,” and check “Verify Flash after Programming.” Click “Program” in the “Flash (128KB)” section (your chip’s “Code Area”). It should program OK.

You should now have a working ATMEL ATMega1284P chip. Any attempts to access the chip with ATMEL Studio and the AVRISP Mk II beyond this point will result in the 0xC0 error.

Appendix 1: Sample EPROM Cartridge Header Program

Anyone is welcome to design a EPROM cartridge using the new cartridge boards. The sample code below will work with an UberGROM board using only a flash chip, the 512K Red boards, or with the appropriate extensions to the header code, with the 2048K Yellow boards.

The first step is to gather the files you want to incorporate into the module you are building. This header program is expecting to load the program files into the 32K memory space of the TI-99/4A for execution. It does not execute programs from the cartridge space as written.

Join the images together using your favorite Hex Editor software (I personally use Hex Workshop) and write a simple control program along the lines of the example below to use as a cartridge header in the first and last banks of your cartridge. Use the header program to read the user’s game-selection key press, move the selected program to the 32K memory space, and branch to the start of the program. The process is much simpler than writing your own applications or games.

The hardest part of this process is keeping track of all of the files in your new image. It would be a good idea to make up a chart like this so you know where everything is when you start to write the code.

Bank Number	Eprom Addr	Write to	CPU Addr	Start Addr	Contents
1	>0000	>6000			GAMES menu program
2	>2000	>6002	>A000	>A000	Paddleball
3	>4000	>6004	>A000	>A056	Centipede
4	>6000	>6006	>C000	>E02E	Dig Dug

5	>8000	>6008	>E000		Dig Dug
6	>A000	>600A	>A000	>A072	Defender
7	>C000	>600C	>A000	>A086	Donkey Kong
8	>E000	>600E	>C000		Donkey Kong
9	>10000	>6010	>C000	>C020	Jungle Hunt
10	>12000	>6012	>E000		Jungle Hunt
11	>14000	>6014	>C000	>C0FE	Moon Patrol
12	>16000	>6016	>E000		Moon Patrol
13	>18000	>6018	>C000	>E0AE	Ms Pacman
14	>1A000	>601A	>E000		Ms Pacman
15	>1C000	>601C	>A000	>A052	Pacman
16	>1E000	>601E	>C000		Pacman
17	>20000	>6020	>A000	>A060	Pole Position
18	>22000	>6022	>C000		Pole Position
19	>24000	>6024	>C000	>DE3A	Protector
20	>26000	>6026	>E000		Protector
21	>28000	>6028	>A000	>CBDA	Shamus
22	>2A000	>602A	>C000		Shamus
23	>2C000	>602C	>A000	>A000	Parsec
24	>2E000	>602E	>BFFA		Parsec
25	>30000	>6030	>DFF4		Parsec
26	>32000	>6032	>268C		Parsec (length >1180)
27	>34000	>6034	>A000	>B458	TI Invaders
28	>36000	>6036	>BFFA		TI Invaders
29	>38000	>6038	>A000	>A50C	4A Flyer

30	>3A000	>603A	>2FFC	>315C	A-Maze-Ing (length >0C00)
31	>3C000	>603C	>A000		A-Maze-Ing (length >1800)
32	>3E000	>603E	>A00E	>A020	Ambulance
33	>40000	>6040	>2000	>2000	Demon Attack (length >1776)
34	>42000	>6042	>A000		Demon Attack
35	>44000	>6044	>BFFA		Demon Attack
36	>46000	>6046	>DFF4		Demon Attack (length >1500)
37	>48000	>6048	>2FFC	>315C	Hangman (length >0C00)
38	>4A000	>604A	>D000		Hangman (length >1800)
39	>4C000	>604C	>A000	>A542	Hopper
40	>4E000	>604E	>A000	>A000	Jawbreaker
41	>50000	>6050	>BFFA		Jawbreaker
42	>52000	>6052	>A000	>AADE	Midnight Mason
43	>54000	>6054	>BFFA		Midnight Mason (length >0080)
44	>56000	>6056	>A000	>CA82	Moon Mine
45	>58000	>6058	>BFFA		Moon Mine
46	>5A000	>605A	>DFF4		Moon Mine
47	>5C000	>605C	>A024	>A024	Rabbit Trail
48	>5E000	>605E	>CFF8	>CFF8	River Rescue
49	>60000	>6060	>EFF2		River Rescue (length >1014)
50	>62000	>6062	>A000	>DA6E	Tennis
51	>64000	>6064	>BFFA		Tennis
52	>66000	>6066	>A000	>B9A0	TI Toad
53	>68000	>6068	>BFFA		TI Toad
54	>6A000	>606A	>A000	>A00C	Tombstone City

55	>6C000	>606C	>BFFC	>C118	Buck Rogers
56	>6E000	>606E	>DFF8		Buck Rogers
57	>70000	>6070	>2000	>20D6	Blackjack & Poker
58	>72000	>6072	>A000		Blackjack & Poker
59	>74000	>6074	>A000	>A000	Cave Creatures
60	>76000	>6076	>A000	>A000	Blasto
61	>78000	>6078	>BFFA		Blasto (length >0200)
62	>7A000	>607A	>A000	>BD4E	Burgertime
63	>7C000	>607C	>BFFA		Burgertime
64	>7E000	>607E	>A020	>A020	Worm Attack

Here is an Assembly Header program example by Tony Knerr that displays a menu, accepts a menu selection, moves the program to expansion memory, and executes it.

```
* GAME LOADER v1.0
* Gazoo
* 5-6-14
        DEF     START                                   Show where program starts
        DATA    >D820,>07D0,>6000
START   B       @START1                                 Go to actual start of program.
VDPWD   EQU     >8C00
VDPST   EQU     >8802
SCAN    EQU     >000E
GPLWS   EQU     >83E0
VDPWA   EQU     >8C02                                   VDP write address register.
VDPRD   EQU     >8800                                   VDP read data register.
BUFFER  EQU     >0FC0
SOUND1  BYTE    >06,>BF,>DF,>FF,>80,>05,>92,>0A,>01,>9F,>00
H01     BYTE    >01
H03     BYTE    >03
H05     BYTE    >05
        EVEN
ZERO    DATA    >0000                                   Define null character.
START1  LIMI    2
        LWPI    >8300                                   Load workspace
        LI      R0,>0180                                Set VDP register 1
```

```
        BLWP    @VWTR
        LI      R0,>0200                Set VDP register 2
        BLWP    @VWTR
        LI      R0,>030E                Set VDP register 3
        BLWP    @VWTR
         LI     R0,>0401                Set VDP register 4
        BLWP    @VWTR
        LI      R0,>0506                Set VDP register 5
        BLWP    @VWTR
        LI      R0,>0600                Set VDP register 6
        BLWP    @VWTR
        LI      R0,>0717                Set VDP register 7
        BLWP    @VWTR
        LI      R0,>0380
        LI      R1,SVENTN
        LI      R2,>0010
        BLWP    @VMBW
        NOP
        LI      R0,>0800                Load menu character set.
        LI      R1,CHRDAU
        LI      R2,>0400
        BLWP    @VMBW
        NOP
        LI      R0,>0000                Put menu on screen
        LI      R1,MENU
        LI      R2,>0300
        BLWP    @VMBW
        NOP
        LI      R0,>01E0                Set VDP register 1.
        BLWP    @VWTR
        BL      @BEEP                   Sound a beep
        NOP
        MOVB    @H03,@>8374
KLOOP1  BLWP    @KSCAN                  Check for keypress.
        MOVB    @>837C,R0
        JEQ     KLOOP1
        CLR     R1
        MOVB    @>8375,R1
        CI      R1,>0F00                Quit if escape is pressed
        JEQ     SEL00
        CI      R1,>4100                A
        JEQ     SEL01
        CI      R1,>4200                B
        JEQ     SEL02
        CI      R1,>4300                C
        JEQ     SEL03
        CI      R1,>4400                D
        JEQ     SEL04
```

```
CI      R1,>4500                        E
JEQ     SEL05
CI      R1,>4600                        F
JEQ     SEL06
CI      R1,>4700                        G
JEQ     SEL07
CI      R1,>4800                        H
JEQ     SEL08
CI      R1,>4900                        I
JEQ     SEL09
CI      R1,>4A00                        J
JEQ     SEL10
CI      R1,>4B00                        K
JEQ     SEL11
CI      R1,>4C00                        L
JEQ     SEL12
CI      R1,>4D00                        M
JEQ     SEL13
CI      R1,>4E00                        N
JEQ     SEL14
CI      R1,>4F00                        O
JEQ     SEL15
CI      R1,>5000                        P
JEQ     SEL16
CI      R1,>5100                        Q
JEQ     SEL17
CI      R1,>5200                        R
JEQ     SEL18
CI      R1,>5300                        S
JEQ     SEL19
CI      R1,>5400                        T
JEQ     SEL20
CI      R1,>5500                        U
JEQ     SEL21
CI      R1,>5600                        V
JEQ     SEL22
CI      R1,>5700                        W
JEQ     SEL23
CI      R1,>5800                        X
JEQ     SEL24
CI      R1,>5900                        Y
JEQ     SEL25
CI      R1,>5A00                        Z
JEQ     SEL26
CI      R1,>3100                        1
JEQ     SEL27
CI      R1,>3200                        2
JEQ     SEL28
```

```
        CI      R1,>3300                    3
        JEQ     SEL29
        CI      R1,>3400                    4
        JEQ     SEL30
        CI      R1,>3500                    5
        JEQ     SEL31
        CI      R1,>3600                    6
        JEQ     SEL32
        CI      R1,>3700                    7
        JEQ     SEL33
        CI      R1,>3800                    8
        JEQ     SEL34
        B       @KLOOP1
SEL00   BL      @BEEP
        B       @QUIT
SEL01   B       @AA
SEL02   B       @BB
SEL03   B       @CC
SEL04   B       @DD
SEL05   B       @EE
SEL06   B       @FF
SEL07   B       @GG
SEL08   B       @HH
SEL09   B       @II
SEL10   B       @JJ
SEL11   B       @KK
SEL12   B       @LL
SEL13   B       @MM
SEL14   B       @NN
SEL15   B       @OO
SEL16   B       @PP
SEL17   B       @QQ
SEL18   B       @RR
SEL19   B       @SS
SEL20   B       @TT
SEL21   B       @UU
SEL22   B       @VV
SEL23   B       @WW
SEL24   B       @XX
SEL25   B       @YY
SEL26   B       @ZZ
SEL27   B       @A1
SEL28   B       @A2
SEL29   B       @A3
SEL30   B       @A4
SEL31   B       @A5
SEL32   B       @A6
SEL33   B       @A7
```

```
SEL34     B       @A8
*****************************************************************************
QUIT      LIMI    2
          CLR     @>837C
          LWPI    >83E0
          BLWP    @>0000                          Go bye-bye.
*****************************************************************************
BEEP      LI      R1,SOUND1
          LI      R0,BUFFER
          LI      R2,11
          BLWP    @VMBW
          LIMI    0
          LI      R10,BUFFER
          MOV     R10,@>83CC
          SOCB    @H01,@>83FD
          MOVB    @H01,@>83CE
          LIMI    2
SNDLP     MOVB    @>83CE,@>83CE
          JNE     SNDLP
          LIMI    0
          RT
*****************************************************************************
BEPCLR    LI      R1,SOUND1                       Beep.
          LI      R0,BUFFER
          LI      R2,11
          BLWP    @VMBW
          LIMI    0
          LI      R10,BUFFER
          MOV     R10,@>83CC
          SOCB    @H01,@>83FD
          MOVB    @H01,@>83CE
          LIMI    2
SNDLP1    MOVB    @>83CE,@>83CE
          JNE     SNDLP1
          LIMI    0
          LI      R0,>0000                        Clear screen
          LI      R1,SPACE
          LI      R2,>300
          BLWP    @VMBW
          NOP
          LI      R0,>0800                        Load TI standard character set.
          LI      R1,CHRDAT
          LI      R2,>0400
          BLWP    @VMBW
          NOP
          MOVB    @ZERO,@>8374                    Set key unit to 0.
          RT
*****************************************************************************
```

```
RUN     LWPI    >83E0
        LI      R11,>0024
        CLR     @>837C
        B       @>8300
*************************************************************************
*       UTILITIES *
*************************************************************************
SET     DATA    >2000
SAVE    DATA     0
REGS    BSS     32
        EVEN
VMBW    DATA    REGSV,V1
REGSV   DATA    0,0,0,>8C02
        DATA    >8C00,0,>8800,0
        DATA    >4000,>8000,0,0
        DATA    0,0,0,0
V1      MOV     R13,R7
        MOV     *R7+,R0
        MOV     *R7+,R1
        MOV     *R7+,R2
        SWPB    R0
        MOVB    R0,*R3
        SWPB    R0
        SOC     R8,R0
        MOVB    R0,*R3
        NOP
V2      MOVB    *R1+,*R4
        DEC     R2
        JNE     V2
        RTWP
VSBW    DATA    REGSV,V3
V3      MOV     R13,R7
        MOV     *R7+,R0
        SWPB    R0
        MOVB    R0,*R3
        SWPB    R0
        SOC     R8,R0
        MOVB    R0,*R3
        NOP
        MOVB    *R7,*R4
        RTWP
VWTR    DATA    REGSV,V4
V4      MOV     *R13,R0
        SOC     R9,R0
        SWPB    R0
        MOVB    R0,*R3
        SWPB    R0
        MOVB    R0,*R3
```

```
        RTWP
KSCAN   DATA    KSCAN,KSCAN+4
        MOV     *R11,R12
        LWPI    >83E0
        BL      @>000E
        LWPI    KSCAN
        MOV     R12,*R11
        RTWP
***************************************************************
MENU    DATA    >2041,>202D,>2050,>6163,>6D61,>6E20     ' A - Pacman '
        DATA    >2020,>0D20,>4C20,>2D20,>5072,>6F74     '  . L - Prot'
        DATA    >6563,>746F,>7220,>2020,>2042,>202D     'ector    B -'
        DATA    >2050,>6172,>7365,>6320,>2020,>0D20     ' Parsec   . '
        DATA    >4D20,>2D20,>416D,>6275,>6C61,>6E63     'M - Ambulanc'
        DATA    >6520,>2020,>2043,>202D,>2053,>6861     'e    C - Sha'
        DATA    >6D75,>7320,>2020,>0D20,>4E20,>2D20     'mus   . N - '
        DATA    >4D6F,>6F6E,>204D,>696E,>6520,>2020     'Moon Mine   '
        DATA    >2044,>202D,>2048,>6F70,>7065,>7220     ' D - Hopper '
        DATA    >2020,>0D20,>4F20,>2D20,>4365,>6E74     '  . O - Cent'
        DATA    >6970,>6564,>6520,>2020,>2045,>202D     'ipede    E -'
        DATA    >2054,>656E,>6E69,>7320,>2020,>0D20     ' Tennis   . '
        DATA    >5020,>2D20,>4D73,>2050,>6163,>6D61     'P - Ms Pacma'
        DATA    >6E20,>2020,>2046,>202D,>2042,>6C61     'n    F - Bla'
        DATA    >7374,>6F20,>2020,>0D20,>5120,>2D20     'sto   . Q - '
        DATA    >4275,>7267,>6572,>7469,>6D65,>2020     'Burgertime  '
        DATA    >2047,>202D,>2048,>616E,>676D,>616E     ' G - Hangman'
        DATA    >2020,>0D20,>5220,>2D20,>5061,>6464     '  . R - Padd'
        DATA    >6C65,>6261,>6C6C,>2020,>2048,>202D     'leball   H -'
        DATA    >2044,>6967,>2044,>7567,>2020,>0D20     ' Dig Dug  . '
        DATA    >5320,>2D20,>4A61,>7762,>7265,>616B     'S - Jawbreak'
        DATA    >6572,>2020,>2049,>202D,>2054,>4920     'er   I - TI '
        DATA    >546F,>6164,>2020,>0D20,>5420,>2D20     'Toad  . T - '
        DATA    >412D,>4D61,>7A65,>2D49,>6E67,>2020     'A-Maze-Ing  '
        DATA    >204A,>202D,>2034,>4120,>466C,>7965     ' J - 4A Flye'
        DATA    >7220,>0D20,>5520,>2D20,>5449,>2049     'r . U - TI I'
        DATA    >6E76,>6164,>6572,>7320,>204B,>202D     'nvaders  K -'
        DATA    >2044,>6566,>656E,>6465,>7220,>0D20     ' Defender . '
        DATA    >5620,>2D20,>4275,>636B,>2052,>6F67     'V - Buck Rog'
        DATA    >6572,>7320,>0B0B,>0B0B,>0B0B,>0B0B     'ers ........'
        DATA    >0B0B,>0B0B,>0B0B,>0C0B,>0B0B,>0B0B     '............'
        DATA    >0B0B,>0B0B,>0B0B,>0B0B,>0B0B,>0B0B     '............'
        DATA    >2020,>2020,>2057,>202D,>2057,>6F72     '     W - Wor'
        DATA    >6D20,>4174,>7461,>636B,>2020,>2020     'm Attack    '
        DATA    >2020,>2020,>2020,>2020,>2020,>2020     '            '
        DATA    >2058,>202D,>2044,>6F6E,>6B65,>7920     ' X - Donkey '
        DATA    >4B6F,>6E67,>2020,>2020,>2020,>2020     'Kong        '
        DATA    >2020,>2020,>2020,>2020,>2059,>202D     '        Y -'
        DATA    >204A,>756E,>676C,>6520,>4875,>6E74     ' Jungle Hunt'
```

```
        DATA    >2020,>2020,>0102,>0320,>2020,>2020     '    ...    '
        DATA    >2020,>2020,>205A,>202D,>204D,>6F6F     '     Z - Moo'
        DATA    >6E20,>5061,>7472,>6F6C,>2020,>2020     'n Patrol    '
        DATA    >0405,>0620,>2020,>2020,>2020,>2020     '...         '
        DATA    >2031,>202D,>2052,>6976,>6572,>2052     ' 1 - River R'
        DATA    >6573,>6375,>6520,>2020,>0708,>0920     'escue   ... '
        DATA    >2020,>2020,>2020,>2020,>2032,>202D     '        2 -'
        DATA    >2052,>6162,>6269,>7420,>5472,>6169     ' Rabbit Trai'
        DATA    >6C20,>2020,>2020,>2020,>2020,>2020     'l           '
        DATA    >2020,>2020,>2033,>202D,>2044,>656D     '     3 - Dem'
        DATA    >6F6E,>2041,>7474,>6163,>6B20,>2020     'on Attack   '
        DATA    >2020,>2020,>2020,>2020,>2020,>2020     '            '
        DATA    >2034,>202D,>2050,>6F6C,>6520,>506F     ' 4 - Pole Po'
        DATA    >7369,>7469,>6F6E,>2020,>2020,>2020     'sition      '
        DATA    >2020,>2020,>2020,>2020,>2035,>202D     '        5 -'
        DATA    >2054,>6F6D,>6273,>746F,>6E65,>2043     ' Tombstone C'
        DATA    >6974,>7920,>2020,>2020,>2020,>2020     'ity         '
        DATA    >2020,>2020,>2036,>202D,>204D,>6964     '     6 - Mid'
        DATA    >6E69,>6768,>7420,>4D61,>736F,>6E20     'night Mason '
        DATA    >2020,>2020,>2020,>2020,>2020,>2020     '            '
        DATA    >2037,>202D,>2043,>6176,>6520,>4372     ' 7 - Cave Cr'
        DATA    >6561,>7475,>7265,>7320,>2020,>2020     'eatures     '
        DATA    >2020,>2020,>2020,>2020,>2038,>202D     '        8 -'
        DATA    >2042,>6C61,>636B,>6A61,>636B,>2026     ' Blackjack &'
        DATA    >2050,>6F6B,>6572,>2020,>2020,>2020     ' Poker      '
*************************************************************
CHRDAT  DATA    >0000,>0000,>0000,>0000,>0000,>003F
        DATA    >3F30,>3030,>3030,>30F0,>F000,>0000
        DATA    >0000,>00FF,>FF00,>0000,>3030,>303F
        DATA    >3F00,>0000,>0000,>00F0,>F030,>3030
        DATA    >3030,>3030,>3030,>3030,>0000,>0000
        DATA    >0000,>0000,>3C42,>9991,>9942,>3C00
        DATA    >0020,>0018,>241C,>0408,>3C42,>99A1
        DATA    >A199,>423C,>0040,>0020,>2038,>2438
        DATA    >0070,>5070,>4854,>1C14,>0070,>4070
        DATA    >001C,>1010,>0020,>0018,>243C,>2018
        DATA    >0040,>0814,>101C,>1010,>0040,>4040
        DATA    >1824,>2418,>0020,>2020,>2808,>0808
        DATA    >0040,>4058,>2408,>103C,>0040,>4058
        DATA    >2408,>2418,>0000,>0000,>0000,>0000
        DATA    >0040,>405C,>1018,>0418,>0040,>4048
        DATA    >1038,>2418,>0040,>405C,>0408,>1010
        DATA    >0040,>4058,>2418,>2418,>0040,>4058
        DATA    >241C,>0408,>0040,>4040,>1824,>3C24
        DATA    >0040,>4050,>101C,>141C,>0040,>4040
        DATA    >1C10,>101C,>0040,>4444,>041C,>141C
        DATA    >0070,>7070,>7070,>7070,>0040,>4C50
        DATA    >101C,>1010,>0000,>0000,>0000,>0000
```

```
DATA    >0010,>1010,>1010,>0010,>0028,>2828
DATA    >0000,>0000,>0028,>287C,>287C,>2828
DATA    >0038,>5450,>3814,>5438,>0060,>6408
DATA    >1020,>4C0C,>0020,>5050,>2054,>4834
DATA    >0008,>0810,>0000,>0000,>0008,>1020
DATA    >2020,>1008,>0020,>1008,>0808,>1020
DATA    >0000,>2810,>7C10,>2800,>0000,>1010
DATA    >7C10,>1000,>0000,>0000,>0030,>1020
DATA    >0000,>0000,>7C00,>0000,>0000,>0000
DATA    >0000,>3030,>0000,>0408,>1020,>4000
DATA    >0038,>4444,>4444,>4438,>0010,>3010
DATA    >1010,>1038,>0038,>4404,>0810,>207C
DATA    >0038,>4404,>1804,>4438,>0008,>1828
DATA    >487C,>0808,>007C,>4078,>0404,>4438
DATA    >0018,>2040,>7844,>4438,>007C,>0408
DATA    >1020,>2020,>0038,>4444,>3844,>4438
DATA    >0038,>4444,>3C04,>0830,>0000,>3030
DATA    >0030,>3000,>0000,>3030,>0030,>1020
DATA    >0008,>1020,>4020,>1008,>0000,>007C
DATA    >007C,>0000,>0020,>1008,>0408,>1020
DATA    >0038,>4404,>0810,>0010,>0038,>445C
DATA    >545C,>4038,>0038,>4444,>7C44,>4444
DATA    >0078,>2424,>3824,>2478,>0038,>4440
DATA    >4040,>4438,>0078,>2424,>2424,>2478
DATA    >007C,>4040,>7840,>407C,>007C,>4040
DATA    >7840,>4040,>003C,>4040,>5C44,>4438
DATA    >0044,>4444,>7C44,>4444,>0038,>1010
DATA    >1010,>1038,>0004,>0404,>0404,>4438
DATA    >0044,>4850,>6050,>4844,>0040,>4040
DATA    >4040,>407C,>0044,>6C54,>5444,>4444
DATA    >0044,>6464,>544C,>4C44,>007C,>4444
DATA    >4444,>447C,>0078,>4444,>7840,>4040
DATA    >0038,>4444,>4454,>4834,>0078,>4444
DATA    >7850,>4844,>0038,>4440,>3804,>4438
DATA    >007C,>1010,>1010,>1010,>0044,>4444
DATA    >4444,>4438,>0044,>4444,>2828,>1010
DATA    >0044,>4444,>5454,>5428,>0044,>4428
DATA    >1028,>4444,>0044,>4428,>1010,>1010
DATA    >007C,>0408,>1020,>407C,>0038,>2020
DATA    >2020,>2038,>0000,>4020,>1008,>0400
DATA    >0038,>0808,>0808,>0838,>0000,>1028
DATA    >4400,>0000,>0000,>0000,>0000,>007C
DATA    >0000,>2010,>0800,>0000,>0000,>0038
DATA    >447C,>4444,>0000,>0078,>2438,>2478
DATA    >0000,>003C,>4040,>403C,>0000,>0078
DATA    >2424,>2478,>0000,>007C,>4078,>407C
DATA    >0000,>007C,>4078,>4040,>0000,>003C
DATA    >405C,>4438,>0000,>0044,>447C,>4444
```

```
        DATA    >0000,>0038,>1010,>1038,>0000,>0008
        DATA    >0808,>4830,>0000,>0024,>2830,>2824
        DATA    >0000,>0040,>4040,>407C,>0000,>0044
        DATA    >6C54,>4444,>0000,>0044,>6454,>4C44
        DATA    >0000,>007C,>4444,>447C,>0000,>0078
        DATA    >4478,>4040,>0000,>0038,>4454,>4834
        DATA    >0000,>0078,>4478,>4844,>0000,>003C
        DATA    >4038,>0478,>0000,>007C,>1010,>1010
        DATA    >0000,>0044,>4444,>4438,>0000,>0044
        DATA    >4428,>2810,>0000,>0044,>4454,>5428
        DATA    >0000,>0044,>2810,>2844,>0000,>0044
        DATA    >2810,>1010,>0000,>007C,>0810,>207C
        DATA    >0018,>2020,>4020,>2018,>0010,>1010
        DATA    >0010,>1010,>0030,>0808,>0408,>0830
        DATA    >0000,>2054,>0800,>0000,>0046,>4F52
        DATA    >0103,>0303,>0000,>0000,>0000,>0000
************************************************
CHRDAU  DATA    >0000,>0000,>0000,>0000,>0103,>0303
        DATA    >0303,>0303,>FC04,>0505,>0406,>020C
        DATA    >0080,>4040,>8000,>0C12,>FF80,>C040
        DATA    >6038,>1C0E,>1921,>213D,>0505,>05C4
        DATA    >BA8A,>8ABA,>A1A1,>A122,>0301,>0000
        DATA    >0000,>0000,>E231,>1018,>0C07,>0300
        DATA    >4C90,>2040,>4020,>E000,>3C42,>99A1
        DATA    >A199,>423C,>0000,>00FF,>0000,>0000
        DATA    >1010,>10FF,>0000,>0000,>1010,>1010
        DATA    >1010,>1010,>0020,>0018,>243C,>2018
        DATA    >0040,>0814,>101C,>1010,>0040,>4040
        DATA    >1824,>2418,>0020,>2020,>2808,>0808
        DATA    >0040,>4058,>2408,>103C,>0040,>4058
        DATA    >2408,>2418,>0040,>4054,>141C,>0404
        DATA    >0040,>405C,>1018,>0418,>0040,>4048
        DATA    >1038,>2418,>0040,>405C,>0408,>1010
        DATA    >0040,>4058,>2418,>2418,>0040,>4058
        DATA    >241C,>0408,>0040,>4040,>1824,>3C24
        DATA    >0040,>4050,>101C,>141C,>0040,>4040
        DATA    >1C10,>101C,>0040,>4444,>041C,>141C
        DATA    >0070,>7070,>7070,>7070,>0040,>4C50
        DATA    >101C,>1010,>0000,>0000,>0000,>0000
        DATA    >0010,>1010,>1000,>1000,>0028,>2828
        DATA    >0000,>0000,>0028,>7C28,>287C,>2800
        DATA    >0038,>5430,>1854,>3800,>0044,>4C18
        DATA    >3064,>4400,>0020,>5020,>5448,>3400
        DATA    >0008,>1020,>0000,>0000,>0008,>1010
        DATA    >1010,>0800,>0020,>1010,>1010,>2000
        DATA    >0044,>287C,>2844,>0000,>0010,>107C
        DATA    >1010,>0000,>0000,>0000,>0030,>1020
        DATA    >0000,>007C,>0000,>0000,>0000,>0000
```

```
DATA    >0030,>3000,>0004,>0810,>2040,>0000
DATA    >003C,>4C54,>6444,>3800,>0010,>3010
DATA    >1010,>3800,>0038,>4408,>1020,>7C00
DATA    >0038,>4418,>0444,>3800,>0008,>1828
DATA    >487C,>0800,>0078,>4078,>0444,>3800
DATA    >0038,>4078,>4444,>3800,>007C,>0408
DATA    >1020,>2000,>0038,>4438,>4444,>3800
DATA    >0038,>4444,>3C04,>7800,>0000,>3030
DATA    >0030,>3000,>0000,>3030,>0030,>1020
DATA    >0000,>1020,>4020,>1000,>0000,>007C
DATA    >007C,>0000,>0000,>1008,>0408,>1000
DATA    >0038,>4408,>1000,>1000,>0038,>4454
DATA    >5840,>3C00,>0038,>4444,>7C44,>4400
DATA    >0078,>2438,>2424,>7800,>0038,>4440
DATA    >4044,>3800,>0078,>2424,>2424,>7800
DATA    >007C,>4078,>4040,>7C00,>007C,>4078
DATA    >4040,>4000,>0038,>4440,>4C44,>3800
DATA    >0044,>447C,>4444,>4400,>0038,>1010
DATA    >1010,>3800,>0004,>0404,>0444,>3800
DATA    >0044,>4850,>7048,>4400,>0040,>4040
DATA    >4040,>7C00,>0044,>6C54,>4444,>4400
DATA    >0044,>6454,>544C,>4400,>0038,>4444
DATA    >4444,>3800,>0078,>4444,>7840,>4000
DATA    >0038,>4444,>544C,>3C00,>0078,>4444
DATA    >7848,>4400,>0038,>4430,>0844,>3800
DATA    >007C,>1010,>1010,>1000,>0044,>4444
DATA    >4444,>3800,>0044,>4444,>4428,>1000
DATA    >0044,>4444,>5454,>2800,>0044,>2810
DATA    >1028,>4400,>0044,>4428,>1010,>1000
DATA    >007C,>0810,>2040,>7C00,>0038,>2020
DATA    >2020,>3800,>0000,>4020,>1008,>0400
DATA    >0038,>0808,>0808,>3800,>0010,>3854
DATA    >1010,>1000,>0000,>0000,>0000,>7C00
DATA    >0020,>1008,>0000,>0000,>0000,>0038
DATA    >4848,>3C00,>0020,>2038,>2424,>3800
DATA    >0000,>001C,>2020,>1C00,>0004,>041C
DATA    >2424,>1C00,>0000,>001C,>2830,>1C00
DATA    >000C,>1038,>1010,>1000,>0000,>001C
DATA    >241C,>0438,>0020,>2038,>2424,>2400
DATA    >0010,>0030,>1010,>3800,>0008,>0008
DATA    >0808,>4830,>0020,>2024,>3828,>2400
DATA    >0030,>1010,>1010,>3800,>0000,>0078
DATA    >5454,>5400,>0000,>0038,>2424,>2400
DATA    >0000,>0018,>2424,>1800,>0000,>0038
DATA    >2438,>2020,>0000,>001C,>241C,>0404
DATA    >0000,>0028,>3420,>2000,>0000,>001C
DATA    >300C,>3800,>0010,>1038,>1010,>0C00
DATA    >0000,>0024,>2424,>1C00,>0000,>0044
```

```
            DATA    >2828,>1000,>0000,>0044,>5454,>2800
            DATA    >0000,>0024,>1818,>2400,>0000,>0024
            DATA    >241C,>0438,>0000,>003C,>0810,>3C00
            DATA    >000C,>1010,>2010,>100C,>0010,>1010
            DATA    >0010,>1010,>0060,>1010,>0810,>1060
            DATA    >0000,>2054,>0800,>0000,>0000,>0000
            DATA    >0000,>0000,>0000,>0000,>0000,>0000
*******************************************************
SPACE       DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
```

```
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
            DATA    >2020,>2020,>2020,>2020,>2020,>2020,>2020,>2020
*********************************************************
SVENTN      DATA    >1717,>1717,>1717,>1717,>1717,>1717,>1717,>1717
THIRTN      DATA    >1313,>1313,>1313,>1313,>1313,>1313,>1313,>1313
*********************************************************
AA          BL      @BEPCLR                                 * Pacman
            LWPI    >83E0
            LI      R1,AADATA
            LI      R2,>8300
            LI      R3,>003A
LOOPAA      MOV     *R1+,*R2+
            DECT    R3
            JGT     LOOPAA
            B       @RUN
AADATA      DATA    >D820,>07D0,>601C,>0201,>6000,>0202
            DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
            DATA    >D820,>07D0,>601E,>0201,>6000,>0202
            DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
            DATA    >D820,>07D0,>6000,>0460,>A052
*********************************************************
BB          BL      @BEPCLR                                 * Parsec
            LWPI    >83E0
            LI      R1,BBDATA
            LI      R2,>8300
            LI      R3,>006A
LOOPBB      MOV     *R1+,*R2+
            DECT    R3
            JGT     LOOPBB
            B       @RUN
BBDATA      DATA    >D820,>07D0,>602C,>0201,>6000,>0202
            DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
            DATA    >D820,>07D0,>602E,>0201,>6000,>0202
            DATA    >BFFA,>0203,>2000,>CCB1,>0643,>15FD
            DATA    >D820,>07D0,>6030,>0201,>6000,>0202
            DATA    >DFF4,>0203,>2000,>CCB1,>0643,>15FD
            DATA    >D820,>07D0,>6032,>0201,>6000,>0202
            DATA    >268C,>0203,>1180,>CCB1,>0643,>15FD
            DATA    >D820,>07D0,>6000,>0460,>A000
*********************************************************
CC          BL      @BEPCLR                                 * Shamus
            LWPI    >83E0
```

```
        LI      R1,CCDATA
        LI      R2,>8300
        LI      R3,>003A
LOOPCC  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPCC
        B       @RUN
CCDATA  DATA    >D820,>07D0,>6028,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>602A,>0201,>6000,>0202
        DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>CBDA
*****************************************************
DD      BL      @BEPCLR                         * Hopper
        LWPI    >83E0
        LI      R1,DDDATA
        LI      R2,>8300
        LI      R3,>0022
LOOPDD  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPDD
        B       @RUN
DDDATA  DATA    >D820,>07D0,>604C,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>A542
*****************************************************
EE      BL      @BEPCLR                         * Tennis
        LI      R0,>0713                        Set VDP register 7
        BLWP    @VWTR
        LI      R0,>0380                        SET COLOR >13
        LI      R1,THIRTN
        LI      R2,>0010
        BLWP    @VMBW
        NOP
        LWPI    >83E0
        LI      R1,EEDATA
        LI      R2,>8300
        LI      R3,>003A
LOOPEE  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPEE
        B       @RUN
EEDATA  DATA    >D820,>07D0,>6062,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6064,>0201,>6000,>0202
        DATA    >BFFA,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>DA6E
*****************************************************
```

```
FF        BL      @BEPCLR                                  * Blasto
          LWPI    >83E0
          LI      R1,FFDATA
          LI      R2,>8300
          LI      R3,>003A
LOOPFF    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPFF
          B       @RUN
FFDATA    DATA    >D820,>07D0,>6076,>0201,>6000,>0202
          DATA    >2000,>0203,>1C00,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6078,>0201,>6000,>0202
          DATA    >A000,>0203,>1800,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>20D6
*****************************************************
GG        LWPI    >83E0                                    * Hangman
          LI      R1,GGDATA
          LI      R2,>8300
          LI      R3,>003A
LOOPGG    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPGG
          CLR     @>A000
          B       @RUN
GGDATA    DATA    >D820,>07D0,>603A,>0201,>6000,>0202
          DATA    >2FFC,>0203,>0C00,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>604A,>0201,>6000,>0202
          DATA    >D000,>0203,>1800,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>315C
*****************************************************
HH        BL      @BEPCLR                                  * Dig Dug
          LWPI    >83E0
          LI      R1,HHDATA
          LI      R2,>8300
          LI      R3,>003A
LOOPHH    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPHH
          B       @RUN
HHDATA    DATA    >D820,>07D0,>6006,>0201,>6000,>0202
          DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6008,>0201,>6000,>0202
          DATA    >E000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>E02E
*****************************************************
II        BL      @BEPCLR                                  * TI Toad
          LWPI    >83E0
          LI      R1,IIDATA
```

```
        LI      R2,>8300
        LI      R3,>003A
LOOPII  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPII
        B       @RUN
IIDATA  DATA    >D820,>07D0,>6066,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6068,>0201,>6000,>0202
        DATA    >BFFA,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>B9A0
*****************************************************
JJ      BL      @BEPCLR                         * 4A Flyer
        LWPI    >83E0
        LI      R1,JJDATA
        LI      R2,>8300
        LI      R3,>0022
LOOPJJ  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPJJ
        B       @RUN
JJDATA  DATA    >D820,>07D0,>6038,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>A50C
*****************************************************
KK      BL      @BEPCLR                         * Defender
        LWPI    >83E0
        LI      R1,KKDATA
        LI      R2,>8300
        LI      R3,>0022
LOOPKK  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPKK
        B       @RUN
KKDATA  DATA    >D820,>07D0,>600A,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>A072
*****************************************************
LL      BL      @BEPCLR                         * Protector
        LWPI    >83E0
        LI      R1,LLDATA
        LI      R2,>8300
        LI      R3,>003A
LOOPLL  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPLL
        B       @RUN
LLDATA  DATA    >D820,>07D0,>6024,>0201,>6000,>0202
```

```
        DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6026,>0201,>6000,>0202
        DATA    >E000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>DE3A
*****************************************************
MM      BL      @BEPCLR                             * Ambulance
        LWPI    >83E0
        LI      R1,MMDATA
        LI      R2,>8300
        LI      R3,>0022
LOOPMM  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPMM
        B       @RUN
MMDATA  DATA    >D820,>07D0,>603E,>0201,>6000,>0202
        DATA    >A00E,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>A020
*****************************************************
NN      BL      @BEPCLR                             * Moon Mine
        LWPI    >83E0
        LI      R1,NNDATA
        LI      R2,>8300
        LI      R3,>0052
LOOPNN  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPNN
        B       @RUN
NNDATA  DATA    >D820,>07D0,>6056,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6058,>0201,>6000,>0202
        DATA    >BFFA,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>605A,>0201,>6000,>0202
        DATA    >DFF4,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>CA82
*****************************************************
OO      BL      @BEPCLR                             * Centipede
        LWPI    >83E0
        LI      R1,OODATA
        LI      R2,>8300
        LI      R3,>0022
LOOPOO  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPOO
        B       @RUN
OODATA  DATA    >D820,>07D0,>6004,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>A056
*****************************************************
```

```
PP        BL      @BEPCLR                                   * Ms Pacman
          LWPI    >83E0
          LI      R1,PPDATA
          LI      R2,>8300
          LI      R3,>003A
LOOPPP    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPPP
          B       @RUN
PPDATA    DATA    >D820,>07D0,>6018,>0201,>6000,>0202
          DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>601A,>0201,>6000,>0202
          DATA    >E000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>E0AE
*******************************************************
QQ        BL      @BEPCLR                                   * Burgertime
          LWPI    >83E0
          LI      R1,QQDATA
          LI      R2,>8300
          LI      R3,>003A
LOOPQQ    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPQQ
          B       @RUN
QQDATA    DATA    >D820,>07D0,>607A,>0201,>6000,>0202
          DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>607C,>0201,>6020,>0202
          DATA    >BFFA,>0203,>1FE0,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>BD4E
*******************************************************
RR        BL      @BEPCLR                                   * Paddleball
          LWPI    >83E0
          LI      R1,RRDATA
          LI      R2,>8300
          LI      R3,>0022
LOOPRR    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPRR
          B       @RUN
RRDATA    DATA    >D820,>07D0,>6002,>0201,>6020,>0202
          DATA    >A000,>0203,>1FE0,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>A000
*******************************************************
SS        BL      @BEPCLR                                   * Jawbreaker
          LWPI    >83E0
          LI      R1,SSDATA
          LI      R2,>8300
          LI      R3,>003A
```

```
LOOPSS   MOV    *R1+,*R2+
         DECT   R3
         JGT    LOOPSS
         B      @RUN
SSDATA   DATA   >D820,>07D0,>604E,>0201,>6000,>0202
         DATA   >A000,>0203,>2000,>CCB1,>0643,>15FD
         DATA   >D820,>07D0,>6050,>0201,>6000,>0202
         DATA   >BFFA,>0203,>2000,>CCB1,>0643,>15FD
         DATA   >D820,>07D0,>6000,>0460,>A000
*******************************************************
TT       BL     @BEPCLR                          * A-Maze-Ing
         LWPI   >83E0
         LI     R1,TTDATA
         LI     R2,>8300
         LI     R3,>003A
LOOPTT   MOV    *R1+,*R2+
         DECT   R3
         JGT    LOOPTT
         CLR    @>D000
         B      @RUN
TTDATA   DATA   >D820,>07D0,>603A,>0201,>6000,>0202
         DATA   >2FFC,>0203,>0C00,>CCB1,>0643,>15FD
         DATA   >D820,>07D0,>603C,>0201,>6000,>0202
         DATA   >A000,>0203,>1800,>CCB1,>0643,>15FD
         DATA   >D820,>07D0,>6000,>0460,>315C
*******************************************************
UU       BL     @BEPCLR                          * TI Invaders
         LWPI   >83E0
         LI     R1,UUDATA
         LI     R2,>8300
         LI     R3,>003A
LOOPUU   MOV    *R1+,*R2+
         DECT   R3
         JGT    LOOPUU
         B      @RUN
UUDATA   DATA   >D820,>07D0,>6034,>0201,>6000,>0202
         DATA   >A000,>0203,>2000,>CCB1,>0643,>15FD
         DATA   >D820,>07D0,>6036,>0201,>6000,>0202
         DATA   >BFFA,>0203,>2000,>CCB1,>0643,>15FD
         DATA   >D820,>07D0,>6000,>0460,>B458
*******************************************************
VV       BL     @BEPCLR                          * Buck Rogers
         LWPI   >83E0
         LI     R1,VVDATA
         LI     R2,>8300
         LI     R3,>003A
LOOPVV   MOV    *R1+,*R2+
         DECT   R3
```

```
        JGT     LOOPVV
        B       @RUN
VVDATA  DATA    >D820,>07D0,>606C,>0201,>6000,>0202
        DATA    >BFFC,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>606E,>0201,>6000,>0202
        DATA    >DFF8,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>C118
*******************************************************
WW      BL      @BEPCLR                         * Worm Attack
        LWPI    >83E0
        LI      R1,WWDATA
        LI      R2,>8300
        LI      R3,>003A
LOOPWW  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPWW
        B       @RUN
WWDATA  DATA    >D820,>07D0,>607E,>0201,>6020,>0202
        DATA    >A020,>0203,>1FE0,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000
        DATA    >0201,>4432,>C801,>C000
        DATA    >0201,>4230,>C801,>C002
        DATA    >0201,>32B0,>C801,>C004
        DATA    >0460,>A020
*******************************************************
XX      BL      @BEPCLR                         * Donkey Kong
        LWPI    >83E0
        LI      R1,XXDATA
        LI      R2,>8300
        LI      R3,>003A
LOOPXX  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPXX
        B       @RUN
XXDATA  DATA    >D820,>07D0,>600C,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>600E,>0201,>6000,>0202
        DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>A086
*******************************************************
YY      BL      @BEPCLR                         * Jungle Hunt
        LI      R0,>0713                        Set VDP register 7
        BLWP    @VWTR
        LWPI    >83E0
        LI      R1,YYDATA
        LI      R2,>8300
        LI      R3,>003A
LOOPYY  MOV     *R1+,*R2+
```

```
        DECT    R3
        JGT     LOOPYY
        B       @RUN
YYDATA  DATA    >D820,>07D0,>6010,>0201,>6000,>0202
        DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6012,>0201,>6000,>0202
        DATA    >E000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>C020
*******************************************************
ZZ      BL      @BEPCLR                         * Moon Patrol
        LWPI    >83E0
        LI      R1,ZZDATA
        LI      R2,>8300
        LI      R3,>003A
LOOPZZ  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPZZ
        B       @RUN
ZZDATA  DATA    >D820,>07D0,>6014,>0201,>6000,>0202
        DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6016,>0201,>6000,>0202
        DATA    >E000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>C0FE
*******************************************************
A1      BL      @BEPCLR                         * River Rescue
        LWPI    >83E0
        LI      R1,A1DATA
        LI      R2,>8300
        LI      R3,>003A
LOOPA1  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPA1
        B       @RUN
A1DATA  DATA    >D820,>07D0,>605E,>0201,>6000,>0202
        DATA    >CFF8,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6060,>0201,>6000,>0202
        DATA    >EFF2,>0203,>1014,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>CFF8
*******************************************************
A2      BL      @BEPCLR                         * Rabbit Trail
        LWPI    >83E0
        LI      R1,A2DATA
        LI      R2,>8300
        LI      R3,>0010
LOOPA2  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPA2
        B       @RUN
```

```
A2DATA    DATA    >D820,>07D0,>605C,>0460,>6024
*****************************************************
A3        BL      @BEPCLR                          * Demon Attack
          LWPI    >83E0
          LI      R1,A3DATA
          LI      R2,>8300
          LI      R3,>006A
LOOPA3    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPA3
          B       @RUN
A3DATA    DATA    >D820,>07D0,>6040,>0201,>6000,>0202
          DATA    >2000,>0203,>1776,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6042,>0201,>6000,>0202
          DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6044,>0201,>6000,>0202
          DATA    >BFFA,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6046,>0201,>6000,>0202
          DATA    >DFF4,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>2000
*****************************************************
A4        BL      @BEPCLR                          * Pole Position
          LWPI    >83E0
          LI      R1,A4DATA
          LI      R2,>8300
          LI      R3,>003A
LOOPA4    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPA4
          B       @RUN
A4DATA    DATA    >D820,>07D0,>6020,>0201,>6000,>0202
          DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6022,>0201,>6000,>0202
          DATA    >C000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>A060
*****************************************************
A5        BL      @BEPCLR                          * Tombstone City
          LWPI    >83E0
          LI      R1,A5DATA
          LI      R2,>8300
          LI      R3,>0022
LOOPA5    MOV     *R1+,*R2+
          DECT    R3
          JGT     LOOPA5
          B       @RUN
A5DATA    DATA    >D820,>07D0,>606A,>0201,>6000,>0202
          DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
          DATA    >D820,>07D0,>6000,>0460,>A00C
```

```
*******************************************************
A6      BL      @BEPCLR                                 * Midnight Mason
        LWPI    >83E0
        LI      R1,A6DATA
        LI      R2,>8300
        LI      R3,>003A
LOOPA6  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPA6
        B       @RUN
A6DATA  DATA    >D820,>07D0,>6052,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6054,>0201,>6000,>0202
        DATA    >BFFA,>0203,>0080,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>AADE
*******************************************************
A7      BL      @BEPCLR                                 * Cave Creatures
        LWPI    >83E0
        LI      R1,A7DATA
        LI      R2,>8300
        LI      R3,>0022
LOOPA7  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPA7
        B       @RUN
A7DATA  DATA    >D820,>07D0,>6074,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>A000
*******************************************************
A8      BL      @BEPCLR                                 * Blackjack & Poker
        LWPI    >83E0
        LI      R1,A8DATA
        LI      R2,>8300
        LI      R3,>003A
LOOPA8  MOV     *R1+,*R2+
        DECT    R3
        JGT     LOOPA8
        B       @RUN
A8DATA  DATA    >D820,>07D0,>6070,>0201,>6000,>0202
        DATA    >2000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6072,>0201,>6000,>0202
        DATA    >A000,>0203,>2000,>CCB1,>0643,>15FD
        DATA    >D820,>07D0,>6000,>0460,>20D6
*******************************************************
        TEXT    '       Gazoo       '
        END
```