UNIVERSITY OF FLORIDA

EEL 4914 Senior Design

Gator µProcessor

Spring 2007

Submitted by:
Kevin Phillipson

Project Abstract

The Gator microprocessor or GμP is a central processing unit to be used for education and research at the University of Florida. This processor will be realized on a development board that will be constructed in the course of this project. The board will contain a programmable gate array, in this case a FPGA. Using this FPGA we can dynamically build and test the CPU by describing and synthesizing it using a hardware description language. The processor will be instruction set & machine code compatible with the Motorola/Freescale 68xx microprocessors. This will allow us to use the extensive library of compliers, assemblers and other tools already available.

Introduction

The ultimate goal is to create a tool which could be used to bridge between Microprocessor Applications (EEL4744C) and Digital Design (EEL4712C) while enhancing both classes. Currently, the courses implement two separate boards. EEL4744C uses a board based on the Freescale 68HC12 micro-controller (Figure 1). It is supported by an EEPROM containing a monitor program, a 4MHz crystal oscillator, a serial port connection, an Altera CPLD, bus drivers and various supporting resistors and capacitors. Most devices are through-hole mounted. EEL4712C uses the BT-U board produced by Binary Technologies which is based on an Altera Cyclone FPGA (Figure 2). The board also features VGA & PS2 interfaces, switch banks and LED displays. The board comes pre-assembled.

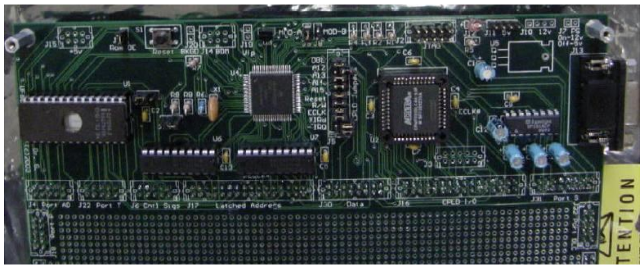


Figure 1: Current 4744 board

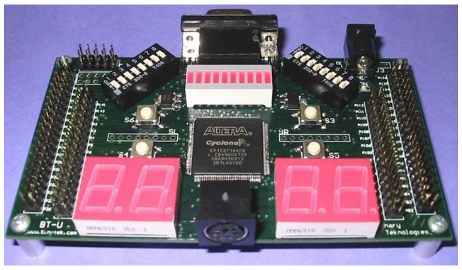


Figure 2: Current 4712 board

The GμP would be a bridge between these two designs, implementing a 68xx compatible CPU core in an Altera Cyclone II FPGA. The new development board would feature an inexpensive wire wrap expansion board to make it flexible enough to be used for both EEL4744 & EEL4712. We believe this approach would allow for a larger understanding of the behavior and design of microprocessors. It would also be a less expensive solution by allowing a single board to be used for multiple courses within the engineering college.

Technical Objectives

The main of objective of the project is to build and test an opencore microprocessor suitable for use in EEL4744. In order to demonstrate the GμP microprocessor design we will produce a development board that can be used for both EEL4744 & EEL4712. A wire wrap board will also be produced to meet the requirement of EEL4744. The block diagram below shows the layout of the development board (Figure 3).

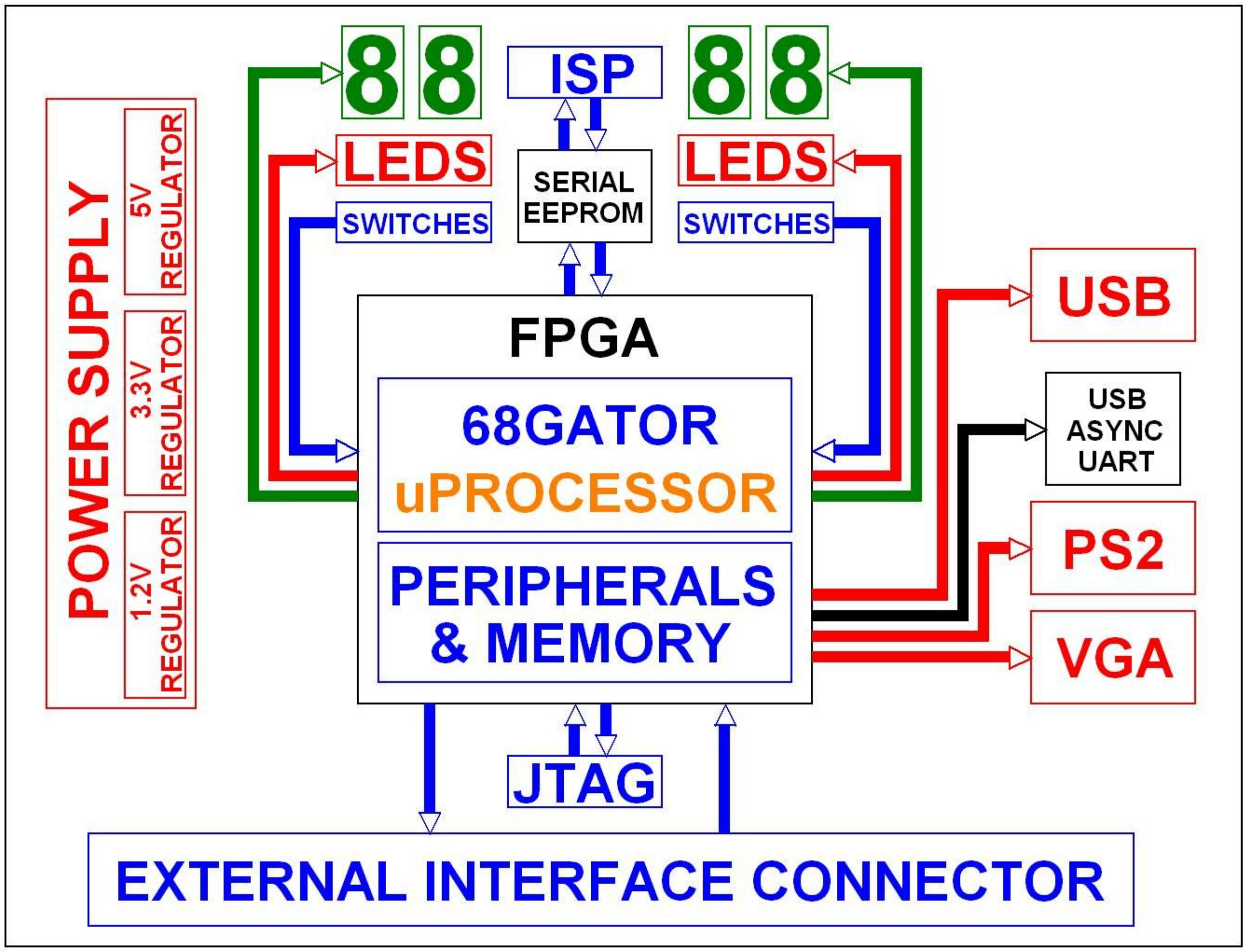
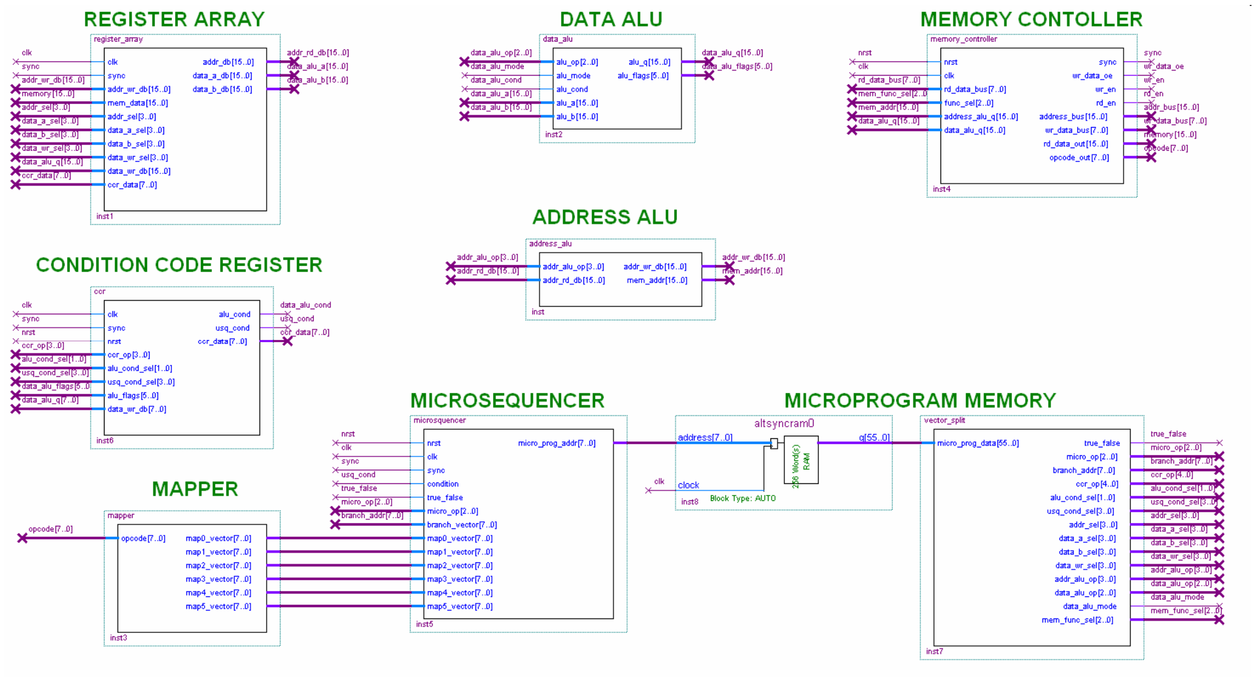


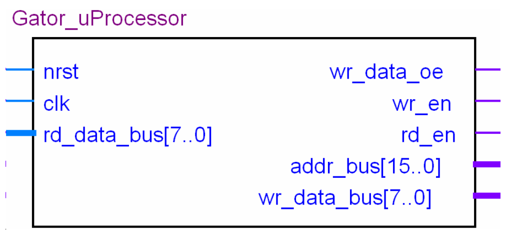
Figure 3: Development Board Block Diagram

The GμP will be a high performance processor which implements a 68xx instruction set. This will be realized by taking advantage of wider internal busses, a faster ALU, improved instruction queue and possibly pipelining. The GμP will be 100% machine code compatible with a Motorola/Freescale 68xx instruction set, but will be a “clean room” design. In other words, the design of the processor will completely independent of Freescale’s design. We are only concerned about implementing the same behavior. Beyond that the GμP will be designed to be as fast and efficient as possible. Our approach is the same as the process the microprocessor industry uses when creating a compatible CPU clones.

Architecture

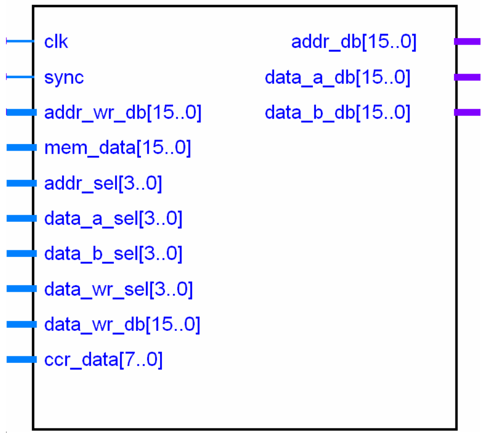


The above diagram shows the internal architecture of the Gator uProcessor. The modules shown above are symbols each linked to a VHDL design file. Each module can be compiled, synthesized and simulated separately. The same is true for the Gator uProcessor itself as shown below.



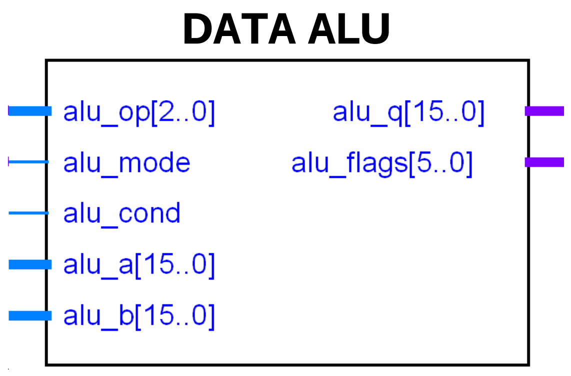
This makes the Gator uProcessor a very powerful tool that an engineer can utilize in many different ways. It can be integrated into a custom System-On-Chip or SoC design. Students can look inside the microprocessor to gain a better understanding of how it works. The following pages describe the individual modules and how they function.

REGISTER ARRAY



Inputs	**Width**	**Description**
clk	1	System clock signal
sync	1	Micro operation synchronization signal
addr_wr_db	16	Address ALU write data bus
mem_data	16	Memory operation data signal
addr_sel	4	Address ALU bus select
data_a_sel	4	Data ALU A bus select
data_b_sel	4	Data ALU B bus select
data_wr_sel	4	Data Register write select
data_wr_db	16	Data ALU write data bus
ccr_data	8	Condition Code Register data signal
Outputs	**Width**	**Description**
addr_db	16	Address ALU data bus
data_a_db	16	Data ALU A data bus
data_b_db	16	Data ALU B data bus

The register array contains the internal registers of the GuP. These include the registers visible in the programmer's model as well as an extra register for the effective address for a particular opcode's addressing mode. The condition code register and the memory controller's data register are accessible through the register array's multiplexers, but are contained separately in their respective modules. The register array has three output busses, two for the data alu inputs and one for the address alu input. The control vectors to select the register operated on are contained in the microcode word.

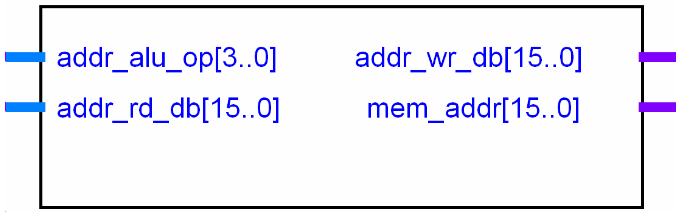


Inputs	**Width**	**Description**
alu_op	3	Data ALU operation select
alu_mode	1	Data ALU 8bit / 16 bit mode select
alu_cond	1	Data ALU condition input
alu_a	16	Data ALU A bus input
alu_b	16	Data ALU B bus input
Outputs	**Width**	**Description**
alu_q	16	Data ALU function output
alu_flags	6	Data ALU condition flag output

The data alu used in the GuP is 16bits wide and is capable of operating on 16bit words in a single clock cycle. The alu can also operate on 8bit data by selecting 8bit mode via the alu mode input. The alu_cond signal provides the carry in bit or shift in bit, depending on the operation selected. The alu implements the full look-ahead carry adder, shifters and logic functions to execute the 68xx instruction set. The output of the alu connects to the memory controller for data write operations and to the register array to write back to the internal registers. The alu also outputs the condition flags from the operation which can modify the condition code register depending on the instruction.

.

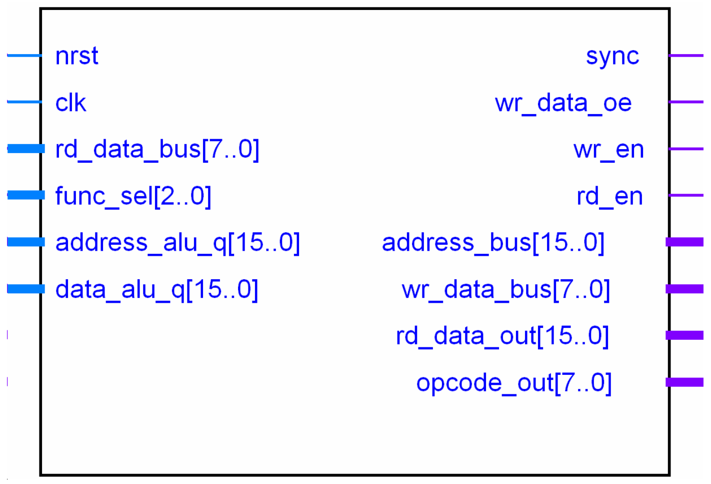
ADDRESS ALU



Inputs	**Width**	**Description**
addr_alu_op	4	Address ALU operation select
addr_rd_db	16	Address ALU input bus
Outputs	**Width**	**Description**
addr_wr_db	16	Address ALU output to Register Array
mem_addr	16	Address ALU output to Memory Controller

The GuP uses a separate alu for address operations. This provides two data paths for alu operations in a single clock cycle. This gives the GuP an advantage over Motorola's single alu design. The alu operations were customized to allow for efficient use of microprogram memory and microcycles. The address alu has two outputs: one connected to the register array for operations on internal registers and one connected to the memory controller to provide the address for data memory operations.

MEMORY CONTROLLER

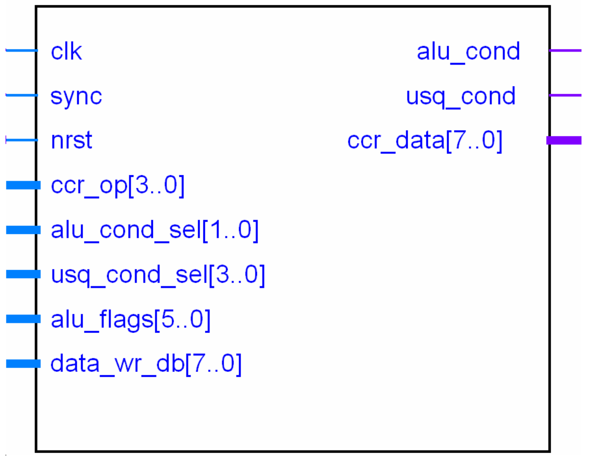


Inputs	**Width**	**Description**
nrst	1	System Reset (negative logic)
clk	1	System Clock
rd_data_bus	8	Microprocessor Read Data Bus
func_sel	3	Memory Controller Function Select
address_alu_q	16	Address ALU function signal
data_alu_q	16	Data ALU function signal
Outputs	**Width**	**Description**
sync	1	Micro operation synchronization signal
wr_data_oe	1	Microprocessor Write Data Output Enable
wr_en	1	Microprocessor Write Enable
rd_en	1	Microprocessor Read Enable
address_bus	16	Microprocessor Address Bus
wr_data_bus	8	Microprocessor Write Data Bus
rd_data_out	16	Memory operation data signal
opcode_out	8	Opcode Register Output

The memory controller is the lowest level state machine in the GuP's hierarchy. Every micro-operation consists of one or more system clock cycles. The number of clock cycles depends on the memory operation. For example, to write a byte to memory takes two clock cycles. If there is no memory access the micro-operation will take a single clock cycle. This is far more efficient than the fixed E-clock cycle used in the Motorola designs. The memory controller outputs a sync signal to all other clock registers in the CPU. This keeps the internal modules in the microprocessor properly synchronized.

Carefully separating the GuP's controller function into two levels simplifies the microcode. The memory controller creates all the necessary timing and control signals to move data in and out of the microprocessor. This simplifies the microcode a great deal. Another advantage of the two levels of hierarchy is the ability to implement different memory interfaces without having to modify the microcode. For example, one could add wait states, a wider 16 bit data bus or a DMA interface by modifying only the memory controller.

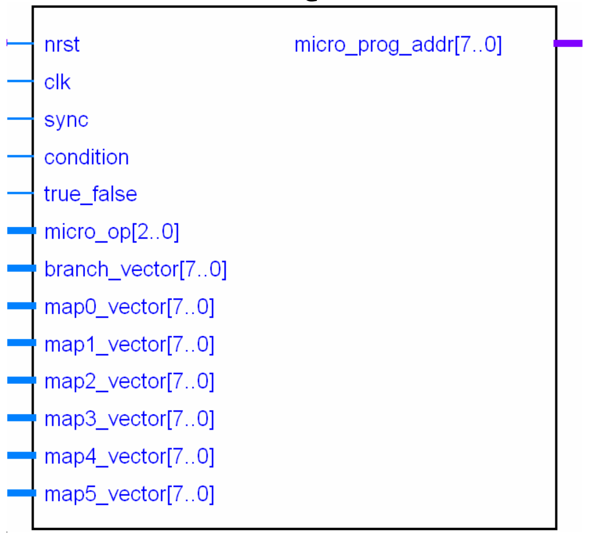
CONDITION CODE REGISTER



Inputs	**Width**	**Description**
clk	1	System Clock
sync	1	Micro operation synchronization signal
nrst	1	System Reset (negative logic)
ccr_op	4	Condition Code Register Operation Select
alu_cond_sel	2	Data ALU condition select
usq_cond_sel	4	Microsequencer condition select
alu_flags	6	Data ALU condition flags
data_wr_db	8	Data ALU write data bus
Outputs	**Width**	**Description**
alu_cond	1	Data ALU condition signal
usq_cond	1	Microsequencer condition signal
ccr_data	8	Condition Code Register Output

The condition code register implements the status register of the 68xx programmer’s model. The register is separated from the register array because it requires special logic to access and modify the register. This includes the logic to mask which bits are modified. As well as output logic for the carry in / shift in of the data alu and the conditional branch of the microsequencer is integrated into this module.

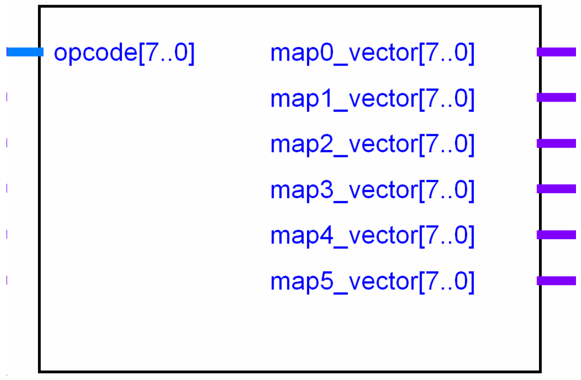
MICROSEQUENCER



Inputs	**Width**	**Description**
nrst	1	System Reset (negative logic)
clk	1	System Clock
sync	1	Micro operation synchronization signal
condition	1	Microsequencer condition input
true_false	1	Microsequencer condition polarity
micro_op	3	Microsequencer opcode
branch_vector	8	Branch vector for JUMP opcode
map0_vector	8	Map vector for JUMP_MAP0
map1_vector	8	Map vector for JUMP_MAP1
map2_vector	8	Map vector for JUMP_MAP2
map3_vector	8	Map vector for JUMP_MAP3
map4_vector	8	Map vector for JUMP_MAP4
map5_vector	8	Map vector for JUMP_MAP5
Outputs	**Width**	**Description**
micro_prog_addr	8	Microprogram memory address

The GuP's controller is realized using a microprogrammed state machine. This method of control is used in many modern CPU designs such as the popular x86 and PowerPC architectures. The microsequencer outputs the address of the current micro operation and decides the address of the next micro operation by evaluating the condition. For more information on the design and use of microprogrammed controllers refer to Microprogrammed State Machine Design by Dr. Michel Lynch. This is book was written before FPGA's were common place, but is still an execellent reference for creating powerful controllers.

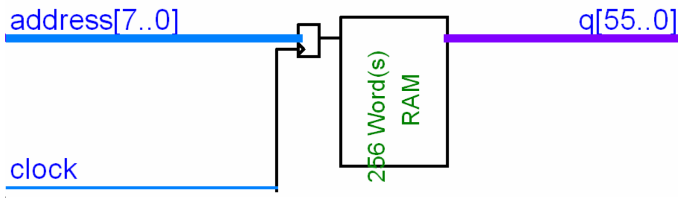
MAPPER



Inputs	**Width**	**Description**
opcode	8	Opcode Register Signal
Outputs	**Width**	**Description**
map0_vector	8	Map vector for JUMP_MAP0
map1_vector	8	Map vector for JUMP_MAP1
map2_vector	8	Map vector for JUMP_MAP2
map3_vector	8	Map vector for JUMP_MAP3
map4_vector	8	Map vector for JUMP_MAP4
map5_vector	8	Map vector for JUMP_MAP5

The mapper is used to decode the opcode. Depending on the opcode input the mapper will output the address mapping structure setup in the microcode. This controls the how the microsequencer implements common sequences for addressing modes of the GuP. The mapper module is automatically generated from the microcode (gucode.asm) using map.c. Map.c takes the gucode.asm file in as an input and generates the mapper.vhd file as an output.

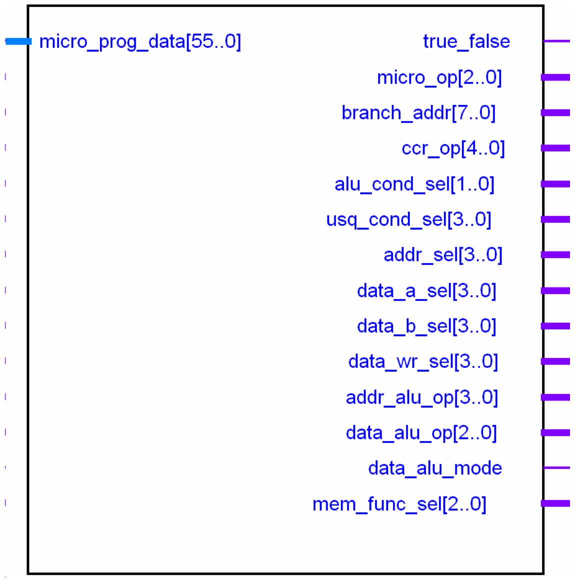
MICROPROGRAM MEMORY



Inputs	Width	Description
address	8	Microprogram memory address
clock	1	System Clock
Outputs	**Width**	**Description**
q	56	Microprogram word

The microprogram memory contains the machine code generated by the microcode macroassembler. A printout of the microcode is provided in the appendix. A full set of tools were created to aid in the development of the microprogram. Information on these tools will be provided in another document.

MICROWORD VECTOR SPLIT

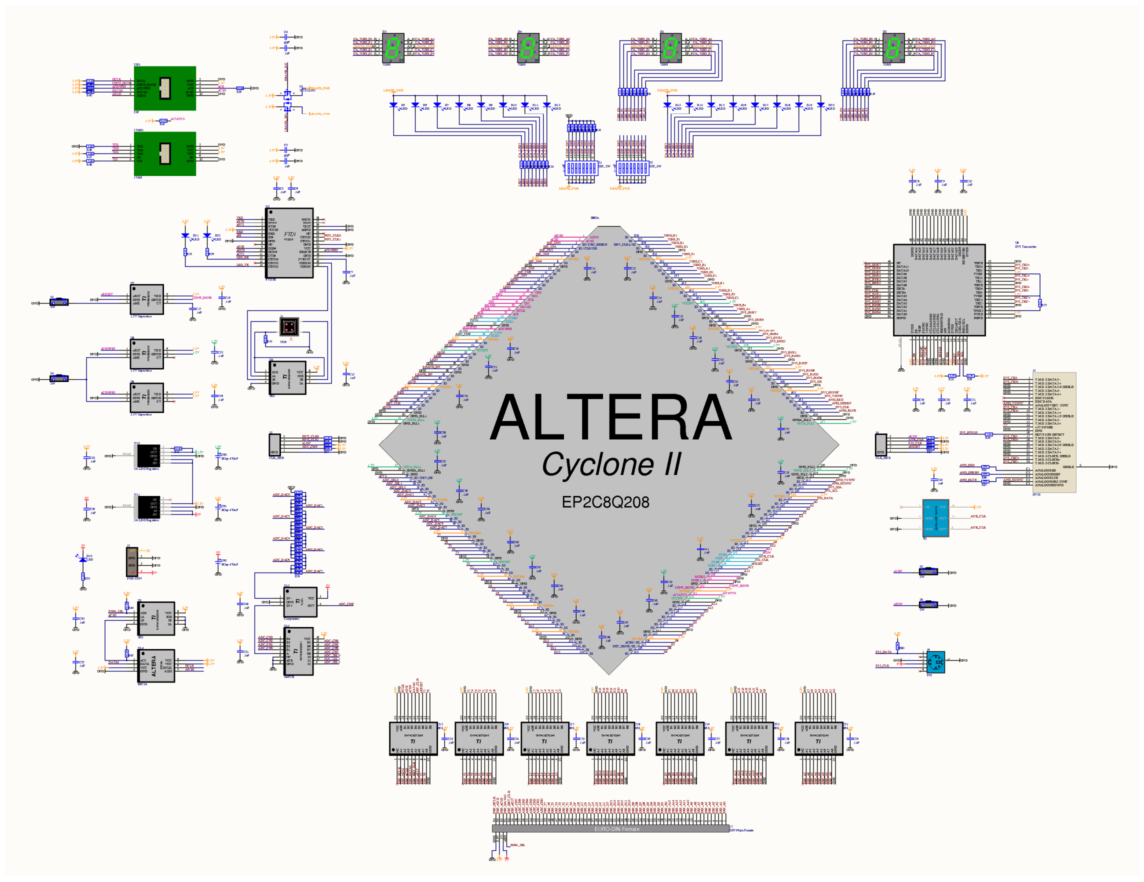


Inputs	**Width**	**Description**
micro_prog_data	56	Microprogram memory data
Outputs	**Width**	**Description**
true_false	1	Microsequencer condition polarity
micro_op	3	Microsequencer opcode
branch_addr	8	Branch vector for JUMP opcode
ccr_op	5	Condition Code Register Operation Select
alu_cond_sel	2	Data ALU condition select
usq_cond_sel	4	Microsequencer condition select
addr_sel	4	Address ALU bus select
data_a_sel	4	Data ALU A bus select
data_b_sel	4	Data ALU B bus select
data_wr_sel	4	Data Register write select
addr_alu_op	4	Address ALU operation select
data_alu_op	3	Data ALU operation select
data_alu_mode	1	Data ALU 8bit / 16 bit mode select
mem_func_sel	3	Memory Controller Function Select

The vector spilt module simply splits the microprogram word into the necessary control vectors.

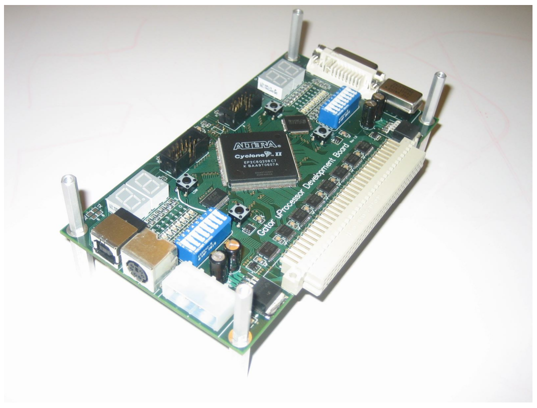
Development Board

The Gator uProcessor Development Board was designed as a direct replacement for the current boards used in EEL4744 and EEL4712. The board measures 6” x 3.25” and is 4 layers. The board was designed in Altium DXP Protel 2004. The board was completely manual routed.



The board components were donated by Altera, Texas Instruments, FTDI and AMP/Tyco. If the board is used by the university these components could be donated to the students, therefore making the board significantly cheaper. This development board is a complex device and will be covered in greater depth in future documentation. Below is a general overview of the major features.

- Altera Cyclone II FPGA
- JTAG Real-time Debugging
- ISP flash configuration ROM
- USB RS232 UART
- DVI/VGA Controller
- PS/2 Interface
- 8-bit, 8-channel ADC
- Flexible user I/O



Conclusion

This project has been a very challenging and rewarding experience. The most difficult aspect of the project was time management. I have worked on a few large projects before, but this was the first time I was responsible not only for the design but also the management side. I also learnt valuable skills in digital design, electronics and PCB design. I am very grateful to the electrical engineering department faculty for providing me with the knowledge and inspiration to take on such a project. Hopefully the Gator uProcessor will help to inspire future students of the University of Florida.

Appendix

```
--==========================> Gator uProccessor <============================--
-- REGISTER_ARRAY.VHD
-- Engineer: Kevin Phillipson
-- Date:     02/09/2007                             Revision:2.09.07
-- Description: Contains the internal registers of the CPU.
--    It also implements the neccessary input and output MUX's.
--============================================================================--


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity register_array is
port
(
   ---------------------------------------> FPGA Clock & Reset
   --
   clk          :  in  std_logic;
   sync         :  in  std_logic;
   --
   --------------------------------------->

   --------------------------------------->
   --
   addr_wr_db      :  in std_logic_vector(15 downto  0);

   mem_data     :  in std_logic_vector(15 downto  0);

   addr_sel     :  in std_logic_vector( 3 downto  0);

   data_a_sel      :  in std_logic_vector( 3 downto  0);
   data_b_sel      :  in std_logic_vector( 3 downto  0);
   data_wr_sel     :  in std_logic_vector( 3 downto  0);

   data_wr_db      :  in std_logic_vector(15 downto  0);

   ccr_data     :  in std_logic_vector( 7 downto  0);

   addr_db         :  out std_logic_vector(15 downto  0);

   data_a_db    :  out std_logic_vector(15 downto  0);
   data_b_db    :  out std_logic_vector(15 downto  0)
   --
   ---------------------------------------<

);
end register_array;

architecture behavior of register_array is

constant  ZERO         :  std_logic_vector( 3 downto 0) := x"0";
constant  EA           :  std_logic_vector( 3 downto 0) := x"1";
constant  PC           :  std_logic_vector( 3 downto 0) := x"2";
constant  SP           :  std_logic_vector( 3 downto 0) := x"3";
constant  Y            :  std_logic_vector( 3 downto 0) := x"4";
constant  X            :  std_logic_vector( 3 downto 0) := x"5";
constant  D            :  std_logic_vector( 3 downto 0) := x"6";
constant  B            :  std_logic_vector( 3 downto 0) := x"7";
constant  A            :  std_logic_vector( 3 downto 0) := x"8";
constant  MEM_U16         :  std_logic_vector( 3 downto 0) := x"9";
constant  MEM_U8       :  std_logic_vector( 3 downto 0) := x"A";
constant  MEM_S8       :  std_logic_vector( 3 downto 0) := x"B";
constant  CCR          :  std_logic_vector( 3 downto 0) := x"C";

signal    a_reg_u8_sig :  std_logic_vector(15 downto 0);
signal    b_reg_u8_sig :  std_logic_vector(15 downto 0);
signal    mem_u16_sig      :  std_logic_vector(15 downto 0);
signal    mem_u8_sig       :  std_logic_vector(15 downto 0);
signal    mem_s8_sig       :  std_logic_vector(15 downto 0);
signal    ccr_sig          :  std_logic_vector(15 downto 0);

signal    ea_reg        :  std_logic_vector(15 downto 0);
signal    pc_reg        :  std_logic_vector(15 downto 0);
signal    sp_reg        :  std_logic_vector(15 downto 0);
signal    y_reg         :  std_logic_vector(15 downto 0);
signal    x_reg         :  std_logic_vector(15 downto 0);
signal    d_reg         :  std_logic_vector(15 downto 0);
signal    b_reg         :  std_logic_vector( 7 downto 0);
signal    a_reg         :  std_logic_vector( 7 downto 0);


begin

process
(
   clk,
   sync,
   data_wr_db,
   addr_wr_db,
   data_wr_sel,
   addr_sel
)
begin

   if (clk'event and clk = '1' and sync = '1') then

      ----------------------------------------> Registers
      --

      ----------------------------------------> A & D Register
      if (data_wr_sel = A) then
         a_reg        <= data_wr_db( 7 downto 0);
      elsif (data_wr_sel = D) then
         a_reg        <= data_wr_db(15 downto 8);
      elsif (addr_sel = A) then
         a_reg        <= addr_wr_db( 7 downto 0);
      elsif (addr_sel = D) then
         a_reg        <= addr_wr_db(15 downto 8);
      end if;

      ----------------------------------------> B & D Register
      if (data_wr_sel = B) then
         b_reg        <= data_wr_db( 7 downto 0);
      elsif (data_wr_sel = D) then
         b_reg        <= data_wr_db( 7 downto 0);
      elsif (addr_sel = B) then
         b_reg        <= addr_wr_db( 7 downto 0);
      elsif (addr_sel = D) then
         b_reg        <= addr_wr_db( 7 downto 0);
      end if;

      ----------------------------------------> X Register
      if (data_wr_sel = X) then
         x_reg        <= data_wr_db;
      elsif (addr_sel = X) then
         x_reg        <= addr_wr_db;
      end if;

      ----------------------------------------> Y Register
      if (data_wr_sel = Y) then
         y_reg        <= data_wr_db;
      elsif (addr_sel = Y) then
         y_reg        <= addr_wr_db;
      end if;

      ----------------------------------------> SP Register
      if (data_wr_sel = SP) then
         sp_reg       <= data_wr_db;
```

```
        elsif (addr_sel = SP) then
            sp_reg        <= addr_wr_db;
        end if;

        ----------------------------------------> PC Register
        if (data_wr_sel = PC) then
            pc_reg        <= data_wr_db;
        elsif (addr_sel = PC) then
            pc_reg        <= addr_wr_db;
        end if;

        ----------------------------------------> EA Register
        if (data_wr_sel = EA) then
            ea_reg        <= data_wr_db;
        elsif (addr_sel = EA) then
            ea_reg        <= addr_wr_db;
        end if;

        --
        ----------------------------------------<

    end if;

end process;



----------------------------------------> Output Logic
--

ccr_sig          <= x"00" & ccr_data;

a_reg_u8_sig <= x"00" & a_reg;
b_reg_u8_sig <= x"00" & b_reg;

d_reg        <= a_reg & b_reg;

mem_u16_sig      <= mem_data;

mem_u8_sig       <= x"00" & mem_data(7 downto 0);

mem_s8_sig       <= mem_data(7) &
                 mem_data(7) &
                 mem_data(7) &
                 mem_data(7) &

                 mem_data(7) &
                 mem_data(7) &
                 mem_data(7) &
                 mem_data(7) &

                 mem_data(7 downto 0);

----------------------------------------> Data A Bus MUX
with data_a_sel select
data_a_db <=
             x"0000"          when ZERO,
             ea_reg       when EA,
             pc_reg       when PC,
             sp_reg       when SP,

             y_reg        when Y,
             x_reg        when X,
             d_reg        when D,
             b_reg_u8_sig when B,

             a_reg_u8_sig when A,
             mem_u16_sig      when MEM_U16,
             mem_u8_sig       when MEM_U8,
             mem_s8_sig       when MEM_S8,
             ccr_sig          when CCR,
             x"0000"          when others;


----------------------------------------> Data B Bus MUX
with data_b_sel select
data_b_db <=
             x"0000"          when ZERO,
             ea_reg       when EA,
             pc_reg       when PC,
             sp_reg       when SP,

             y_reg        when Y,
             x_reg        when X,
             d_reg        when D,
             b_reg_u8_sig when B,

             a_reg_u8_sig when A,
             mem_u16_sig      when MEM_U16,
             mem_u8_sig       when MEM_U8,
             mem_s8_sig       when MEM_S8,
             ccr_sig          when CCR,

             x"0000"          when others;


----------------------------------------> Address Bus MUX
with addr_sel select
addr_db   <=
             x"0000"          when ZERO,
             ea_reg       when EA,
             pc_reg       when PC,
             sp_reg       when SP,

             y_reg        when Y,
             x_reg        when X,
             d_reg        when D,
             b_reg_u8_sig when B,

             a_reg_u8_sig when A,
             mem_u16_sig      when MEM_U16,
             mem_u8_sig       when MEM_U8,
             mem_s8_sig       when MEM_S8,
             ccr_sig          when CCR,

             x"0000"          when others;


end behavior;
```

```
--===========================> Gator uProccessor <============================--
-- DATA_ALU.VHD
-- Engineer: Kevin Phillipson
-- Date:     02/09/2007                              Revision:3.02.07
-- Description:
--
--============================================================================--

library ieee;
use ieee.std_logic_1164.all;

entity data_alu is

--=============================> Port Map <==================================--

port
(
   ---------------------------------------> Inputs
   --
   alu_op    :  in std_logic_vector( 2 downto  0);
   alu_mode  :  in std_logic;

   alu_cond  :  in std_logic;

   alu_a     :  in std_logic_vector(15 downto  0);
   alu_b     :  in std_logic_vector(15 downto  0);
   --
   --------------------------------------->


   ---------------------------------------> Outputs
   --
   alu_q     :  out std_logic_vector(15 downto  0);

   alu_flags :  out std_logic_vector( 5 downto  0)
   --
   ---------------------------------------<

);

--============================================================================--

end data_alu;

architecture behavior of data_alu is

--==============================> Signals <=================================--


---------------------------------------> Operation Select Constants
--
constant  A_PLUS_B        :  std_logic_vector(2 downto 0)  := "000";
constant  A_PLUS_NOT_B    :  std_logic_vector(2 downto 0)  := "001";
constant  A_AND_B         :  std_logic_vector(2 downto 0)  := "010";
constant  A_OR_B          :  std_logic_vector(2 downto 0)  := "011";
constant  A_XOR_B         :  std_logic_vector(2 downto 0)  := "100";
constant  LSHIFT_A        :  std_logic_vector(2 downto 0)  := "101";
constant  RSHIFT_A        :  std_logic_vector(2 downto 0)  := "110";
--
---------------------------------------<

---------------------------------------> Mode Select Constants
--
constant  MODE_8          :  std_logic := '0';
constant  MODE_16         :  std_logic := '1';
--
---------------------------------------<

---------------------------------------> Internal ALU Signals
--
signal    sig_a_plus_b     :  std_logic_vector(15 downto  0);
signal    sig_a_plus_not_b :  std_logic_vector(15 downto  0);
signal    sig_a_and_b         :  std_logic_vector(15 downto  0);
signal    sig_a_or_b          :  std_logic_vector(15 downto  0);
signal    sig_a_xor_b         :  std_logic_vector(15 downto  0);
signal    sig_lshift_a     :  std_logic_vector(15 downto  0);
signal    sig_rshift_a     :  std_logic_vector(15 downto  0);

signal    sig_cin             :  std_logic_vector(15 downto  0);
signal    sig_cout         :  std_logic_vector(15 downto  0);
signal    sig_vout         :  std_logic_vector(15 downto  0);

signal    sig_a            :  std_logic_vector(15 downto  0);
signal    sig_b            :  std_logic_vector(15 downto  0);
signal    sig_q            :  std_logic_vector(15 downto  0);

signal    sig_c8           :  std_logic;   -- 8bit Carry
signal    sig_v8           :  std_logic;   -- 8bit Overflow
signal    sig_z8           :  std_logic;   -- 8bit Zero
signal    sig_n8           :  std_logic;   -- 8bit Negative
signal    sig_h8           :  std_logic;   -- 8bit Half Carry
signal    sig_a8           :  std_logic;   -- 8bit A Input Sign Bit

signal    sig_c16             :  std_logic;   -- 16bit Carry
signal    sig_v16             :  std_logic;   -- 16bit Overflow
signal    sig_z16             :  std_logic;   -- 16bit Zero
signal    sig_n16             :  std_logic;   -- 16bit Negative
signal    sig_h16             :  std_logic;   -- 16bit Half Carry
signal    sig_a16             :  std_logic;   -- 16bit A Input Sign Bit
--
----------------------------------------<

--============================================================================--

begin

--=========================> ALU Architecture <=============================--

----------------------------------------> Input Signals
--
sig_a  <= alu_a;
sig_b  <= (not alu_b) when (alu_op = A_PLUS_NOT_B) else alu_b;
--
----------------------------------------<

----------------------------------------> Boolean Logic
--
sig_a_and_b      <=  sig_a  and sig_b;
sig_a_or_b       <=  sig_a  or  sig_b;
sig_a_xor_b      <=  sig_a  xor sig_b;
--
----------------------------------------<

----------------------------------------> Full Look Ahead Carry Adder
--
sig_cin                <=  sig_cout(14 downto  0) &  alu_cond;
sig_cout            <= sig_a_and_b            or (sig_a_or_b  and sig_cin);
sig_a_plus_b        <= sig_a_xor_b            xor sig_cin;
sig_a_plus_not_b    <= sig_a_plus_b;
sig_vout(15 downto 1)  <=  sig_cout(15 downto 1)  xor sig_cout(14 downto 0);
--
----------------------------------------<

----------------------------------------> Shifters
--
sig_lshift_a           <=  sig_a(14 downto  0) &  alu_cond;

sig_rshift_a(15  downto  8)   <= alu_cond        &  sig_a(15 downto  9);

with alu_mode select
sig_rshift_a( 7 downto  0) <=  alu_cond        &  sig_a( 7 downto  1) when
MODE_8,
                           sig_a( 8)       &  sig_a( 7 downto  1) when others;
--
```

```
----------------------------------------<

----------------------------------------> ALU Operation Output
--
with alu_op select
sig_q  <= sig_a_plus_b     when   A_PLUS_B,
          sig_a_plus_not_b when   A_PLUS_NOT_B,
          sig_a_and_b         when   A_AND_B,
          sig_a_or_b          when   A_OR_B,
          sig_a_xor_b         when   A_XOR_B,
          sig_lshift_a     when   LSHIFT_A,
          sig_rshift_a     when   others;

alu_q  <= sig_q;
--
----------------------------------------<

----------------------------------------> 8bit Flags
--
with alu_op select
sig_c8 <= sig_cout( 7) when   A_PLUS_B,
          not sig_cout( 7) when   A_PLUS_NOT_B,
          sig_a( 7)    when   LSHIFT_A,
          sig_a( 0)    when   RSHIFT_A,
          '0'          when   others;

with alu_op select
sig_v8 <= sig_vout( 7) when   A_PLUS_B,
          sig_vout( 7) when   A_PLUS_NOT_B,
          '0'          when   others;

sig_z8 <= '1' when (sig_q( 7 downto  0) = x"00") else '0';

sig_n8 <= sig_q( 7);

sig_h8 <= sig_cout( 3);

sig_a8 <= sig_a( 7);
--
----------------------------------------<

----------------------------------------> 16bit Flags
--
with alu_op select
sig_c16   <= sig_cout(15) when   A_PLUS_B,
          not sig_cout(15) when   A_PLUS_NOT_B,
          sig_a(15)    when   LSHIFT_A,
          sig_a( 0)    when   RSHIFT_A,
          '0'          when   others;

with alu_op select
sig_v16   <= sig_vout(15) when   A_PLUS_B,
          sig_vout(15) when   A_PLUS_NOT_B,
          '0'          when   others;

sig_z16   <= '1' when (sig_q = x"0000") else '0';

sig_n16   <= sig_q(15);

sig_h16   <= sig_cout( 3);

sig_a16   <= sig_a(15);
--
----------------------------------------<

----------------------------------------> ALU Flag Output
--
alu_flags( 0)<= sig_c8 when (alu_mode = MODE_8) else sig_c16;
alu_flags( 1)<= sig_v8 when (alu_mode = MODE_8) else sig_v16;
alu_flags( 2)<= sig_z8 when (alu_mode = MODE_8) else sig_z16;
alu_flags( 3)<= sig_n8 when (alu_mode = MODE_8) else sig_n16;
alu_flags( 4)<= sig_h8 when (alu_mode = MODE_8) else sig_h16;
alu_flags( 5)<= sig_a8 when (alu_mode = MODE_8) else sig_a16;
--
----------------------------------------<

--===========================================================================--
end behavior;
```

```
--==========================> Gator uProccessor <============================--
-- ADDRESS_ALU.VHD
-- Engineer: Kevin Phillipson
-- Date:     02/09/2007                              Revision:3.02.07
-- Description:
--
--==============================================================================--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity address_alu is

--============================> Port Map <===================================--

port
(
   ----------------------------------------> Inputs
   --
   addr_alu_op  :  in std_logic_vector( 3 downto  0);

   addr_rd_db   :  in std_logic_vector(15 downto  0);
   --
   ---------------------------------------->


   ----------------------------------------> Outputs
   --
   addr_wr_db   :  out std_logic_vector(15 downto  0);

   mem_addr  :  out std_logic_vector(15 downto  0)
   --
   ----------------------------------------<

);

--==============================================================================--

end address_alu;

architecture behavior of address_alu is

--==============================> Signals <==================================--


----------------------------------------> Operation Select Constants
--
constant  PRE_INC     :  std_logic_vector( 3 downto  0)   := "0000";
constant  PRE_INC2  :  std_logic_vector( 3 downto  0)   := "0001";
constant  PRE_DEC     :  std_logic_vector( 3 downto  0)   := "0010";
constant  PRE_DEC2  :  std_logic_vector( 3 downto  0)   := "0011";
constant  POST_INC  :  std_logic_vector( 3 downto  0)   := "0100";
constant  POST_INC2 :  std_logic_vector( 3 downto  0)   := "0101";
constant  POST_DEC  :  std_logic_vector( 3 downto  0)   := "0110";
constant  POST_DEC2 :  std_logic_vector( 3 downto  0)   := "0111";
constant  PASS      :  std_logic_vector( 3 downto  0)   := "1000";
--
----------------------------------------<

constant  POS1      :  std_logic_vector(15 downto  0)   := x"0001";
constant  POS2      :  std_logic_vector(15 downto  0)   := x"0002";
constant  NEG1      :  std_logic_vector(15 downto  0)   := x"FFFF";
constant  NEG2      :  std_logic_vector(15 downto  0)   := x"FFFE";
constant  ZERO      :  std_logic_vector(15 downto  0)   := x"0000";

----------------------------------------> Internal ALU Signals
--
signal    alu_q     :  std_logic_vector(15 downto  0);
signal    alu_b     :  std_logic_vector(15 downto  0);
--
```

```
----------------------------------------<

--==========================================================================--

begin

--=========================> ALU Architecture <==============================--

with addr_alu_op select
alu_b     <=
          POS1      when   PRE_INC,
          POS2      when   PRE_INC2,
          NEG1      when   PRE_DEC,
          NEG2      when   PRE_DEC2,
          POS1      when   POST_INC,
          POS2      when   POST_INC2,
          NEG1      when   POST_DEC,
          NEG2      when   POST_DEC2,
          ZERO      when   others;--PASS

alu_q     <= addr_rd_db + alu_b;

addr_wr_db   <= alu_q;

with addr_alu_op select
mem_addr  <=
          alu_q     when   PRE_INC,
          alu_q     when   PRE_INC2,
          alu_q     when   PRE_DEC,
          alu_q     when   PRE_DEC2,
          addr_rd_db   when   POST_INC,
          addr_rd_db   when   POST_INC2,
          addr_rd_db   when   POST_DEC,
          addr_rd_db   when   POST_DEC2,
          alu_q     when   others;--PASS

--==========================================================================--
end behavior;
```

```
--=========================> Gator uProccessor <===========================--
-- MEMORY_CONTROLLER.VHD
-- Engineer: Kevin Phillipson
-- Date:     02/09/2007                              Revision:2.09.07
-- Description:
--
--========================================================================--


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity memory_controller is
port
(
   --------------------------------------> Clock, Reset & Sync
   --
   nrst              :  in    std_logic;
   clk               :  in    std_logic;
   sync              :  out   std_logic;
   --
   --------------------------------------<

   --------------------------------------> Memory & Control Bus
   --
   wr_data_oe           :  out   std_logic;

   wr_en             :  out   std_logic;
   rd_en             :  out   std_logic;

   address_bus          :  out   std_logic_vector(15 downto  0);

   rd_data_bus          :  in    std_logic_vector( 7 downto  0);
   wr_data_bus          :  out   std_logic_vector( 7 downto  0);
   --
   --------------------------------------<

   --------------------------------------> Internal Architecture Signals
   --
   func_sel          :  in    std_logic_vector( 2 downto  0);

   address_alu_q     :  in    std_logic_vector(15 downto  0);
   data_alu_q           :  in    std_logic_vector(15 downto  0);

   rd_data_out          :  out   std_logic_vector(15 downto  0);
   opcode_out           :  out   std_logic_vector( 7 downto 0)
   --
   --------------------------------------<
);
end memory_controller;


architecture behavior of memory_controller is


--------------------------------------> Signals & Constants
--
type state_type is
(
   mem_cycle_0,
   mem_cycle_1,
   mem_cycle_2,
   mem_cycle_3,
   mem_cycle_4,
   mem_cycle_5
);

signal   state_reg   :  state_type;
signal   state_nxt   :  state_type;
signal    sync_nxt      :  std_logic;
signal    sync_reg      :  std_logic;

signal    address_load :  std_logic;
signal    address_inc      :  std_logic;
signal    address_bus_reg  :  std_logic_vector(15 downto  0);

signal    wr_data_oe_nxt   :  std_logic;
signal    wr_data_oe_reg   :  std_logic;
signal    wr_data_lo_load  :  std_logic;
signal    wr_data_hi_load  :  std_logic;
signal    wr_en_nxt     :  std_logic;
signal    wr_en_reg     :  std_logic;
signal    wr_data_bus_reg  :  std_logic_vector( 7 downto  0);

signal    rd_data_lo_load  :  std_logic;
signal    rd_data_hi_load  :  std_logic;
signal    rd_en_nxt     :  std_logic;
signal    rd_en_reg     :  std_logic;
signal    rd_data_out_reg  :  std_logic_vector(15 downto  0);

signal    opcode_load      :  std_logic;
signal    opcode_out_reg   :  std_logic_vector( 7 downto  0);

constant  IDLE          :  std_logic_vector( 2 downto  0)   :=  "000";
constant  READ_BYTE     :  std_logic_vector( 2 downto  0)   :=  "001";
constant  WRITE_BYTE       :  std_logic_vector( 2 downto  0)   :=  "010";
constant  READ_WORD     :  std_logic_vector( 2 downto  0)   :=  "011";
constant  WRITE_WORD       :  std_logic_vector( 2 downto  0)   :=  "100";
constant  READ_OPCODE      :  std_logic_vector( 2 downto  0)   :=  "101";
--
---------------------------------------<


begin


---------------------------------------> Next State & Output Logic
--
process
(
   state_reg,
   func_sel
)
begin

   ---------------------------------------> Default Values
   --
   address_load <= '0';
   address_inc      <= '0';

   wr_en_nxt    <= '0';
   wr_data_oe_nxt   <= '0';
   wr_data_lo_load  <= '0';
   wr_data_hi_load  <= '0';

   rd_en_nxt    <= '0';
   rd_data_lo_load  <= '0';
   rd_data_hi_load  <= '0';

   opcode_load      <= '0';

   sync_nxt     <= '0';
   state_nxt    <= mem_cycle_0;
   --
   ---------------------------------------<

   case state_reg is

      when mem_cycle_0 =>

         --------------------------------------->
```

```
          --
          if (func_sel = READ_OPCODE) then
             address_load <= '1';         -- address_bus_reg becomes valid
next state & is maintained
             state_nxt    <= mem_cycle_1;
          elsif (func_sel = READ_BYTE) then
             address_load <= '1';         -- address_bus_reg becomes valid
next state & is maintained
             state_nxt    <= mem_cycle_1;
          elsif (func_sel = WRITE_BYTE) then
             address_load <= '1';         -- address_bus_reg becomes valid
next state & is maintained
             wr_data_lo_load  <= '1';         -- wr_data_bus_reg becomes valid
next state & is maintained
             wr_data_oe_nxt   <= '1';         -- wr_data_oe_reg = '1' next
state only
             state_nxt    <= mem_cycle_1;
          elsif (func_sel = READ_WORD) then
             address_load <= '1';         -- address_bus_reg becomes valid
next state & is maintained
             state_nxt    <= mem_cycle_1;
          elsif (func_sel = WRITE_WORD) then
             address_load <= '1';         -- address_bus_reg becomes valid
next state & is maintained
             wr_data_hi_load  <= '1';         -- wr_data_bus_reg becomes valid
next state & is maintained
             wr_data_oe_nxt   <= '1';         -- wr_data_oe_reg = '1' next
state only
             state_nxt    <= mem_cycle_1;
          else
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          end if;
          --
          ---------------------------------------<

      when mem_cycle_1 =>

          --------------------------------------->
          --
          if (func_sel = READ_OPCODE) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_2;
          elsif (func_sel = READ_BYTE) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_2;
          elsif (func_sel = WRITE_BYTE) then
             wr_data_oe_nxt   <= '1';         -- wr_data_oe_reg = '1' next
state only
             wr_en_nxt    <= '1';         -- wr_en_reg = '1' half of this
state and the next only
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          elsif (func_sel = READ_WORD) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_2;
          elsif (func_sel = WRITE_WORD) then
             wr_data_oe_nxt   <= '1';         -- wr_data_oe_reg = '1' next
state only
             wr_en_nxt    <= '1';         -- wr_en_reg = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_2;
          else
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          end if;
          --
          ---------------------------------------<

       when mem_cycle_2 =>

          --------------------------------------->
          --
          if (func_sel = READ_OPCODE) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             opcode_load     <= '1';         -- opcode_out_reg becomes valid
next state & is maintained
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          elsif (func_sel = READ_BYTE) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             rd_data_lo_load <= '1';         -- rd_data_out_reg becomes valid
next state & is maintained
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          elsif (func_sel = READ_WORD) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             rd_data_hi_load <= '1';         -- rd_data_out_reg becomes valid
next state & is maintained
             state_nxt    <= mem_cycle_3;
          elsif (func_sel = WRITE_WORD) then
             address_inc     <= '1';         -- address_bus_reg becomes valid
next state & is maintained
             wr_data_lo_load <= '1';         -- wr_data_bus_reg becomes valid
next state & is maintained
             wr_data_oe_nxt  <= '1';         -- wr_data_oe_reg = '1' next
state only
             state_nxt    <= mem_cycle_3;
          else
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          end if;
          --
          ---------------------------------------<

       when mem_cycle_3 =>

          --------------------------------------->
          --
          if (func_sel = READ_WORD) then
             address_inc     <= '1';         -- address_bus_reg becomes valid
next state & is maintained
             state_nxt    <= mem_cycle_4;
          elsif (func_sel = WRITE_WORD) then
             wr_data_oe_nxt  <= '1';         -- wr_data_oe_reg = '1' next
state only
             wr_en_nxt    <= '1';         -- wr_en_reg = '1' half of this
state and the next only
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          else
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          end if;
          --
          ---------------------------------------<

       when mem_cycle_4 =>

          --------------------------------------->
          --
          if (func_sel = READ_WORD) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_5;
          else
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          end if;
          --
          ----------------------------------------<

       when mem_cycle_5 =>

          ---------------------------------------->
          --
          if (func_sel = READ_WORD) then
             rd_en_nxt    <= '1';         -- rd_en_reg = '1' half of this
state and the next only
             rd_data_lo_load  <= '1';         -- rd_data_out_reg becomes valid
next state & is maintained
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          else
             sync_nxt     <= '1';         -- sync_reg  = '1' half of this
state and the next only
             state_nxt    <= mem_cycle_0;
          end if;
          --
          ----------------------------------------<

    end case;

end process;
--
----------------------------------------<



----------------------------------------> Rising Edge Clocked Registers
--
process
(
   nrst,
   clk,
   state_nxt,
   wr_data_oe_nxt,
   address_load,
   address_alu_q,
   address_inc,
   wr_data_hi_load,
   wr_data_lo_load,
   data_alu_q,
   rd_data_hi_load,
   rd_data_lo_load,
   rd_data_bus,
   opcode_load
)
begin

   if (clk'event and clk = '1') then

      if (nrst = '0') then

         state_reg    <= mem_cycle_0;
         wr_data_oe_reg   <= '0';

      else

         state_reg    <= state_nxt;
```

```
            wr_data_oe_reg   <= wr_data_oe_nxt;

            if (address_load = '1') then
               address_bus_reg  <= address_alu_q;
            elsif (address_inc = '1') then
               address_bus_reg  <= address_bus_reg + '1';
            end if;

            if (wr_data_hi_load = '1') then
               wr_data_bus_reg  <= data_alu_q(15 downto  8);
            elsif (wr_data_lo_load = '1') then
               wr_data_bus_reg  <= data_alu_q( 7 downto  0);
            end if;

            if (rd_data_hi_load = '1') then
               rd_data_out_reg(15 downto  8) <= rd_data_bus;
            end if;

            if (rd_data_lo_load = '1') then
               rd_data_out_reg( 7 downto  0) <= rd_data_bus;
            end if;

            if (opcode_load = '1') then
               opcode_out_reg   <= rd_data_bus;
            end if;

         end if;

      end if;

   end process;
   --
   ----------------------------------------<


   ----------------------------------------> Falling Edge Clocked Registers
   --
   process
   (
      nrst,
      clk,
      wr_en_nxt,
      rd_en_nxt,
      sync_nxt
   )
   begin

      if (clk'event and clk = '0') then

         if (nrst = '0') then

            sync_reg  <= '0';

            wr_en_reg <= '0';
            rd_en_reg <= '0';

         else

            sync_reg  <= sync_nxt;

            wr_en_reg <= wr_en_nxt;
            rd_en_reg <= rd_en_nxt;

         end if;

      end if;

   end process;
   --
   ----------------------------------------<
```

```
----------------------------------------> Outputs
--
sync       <= sync_reg;
rd_en      <= rd_en_reg;
wr_en      <= wr_en_reg;
wr_data_oe   <= wr_data_oe_reg;
address_bus  <= address_bus_reg;
wr_data_bus  <= wr_data_bus_reg;
rd_data_out  <= rd_data_out_reg;
opcode_out   <= opcode_out_reg;
--
----------------------------------------<


end behavior;
```

```
--==========================> Gator uProccessor <===========================--
-- CCR.VHD
-- Engineer: Kevin Phillipson
-- Date:     02/09/2007                              Revision:3.02.07
-- Description:
--
--=============================================================================--

library ieee;
use ieee.std_logic_1164.all;

entity ccr is

--============================> Port Map <===================================--

port
(
   ---------------------------------------> Inputs
   --
   clk          :  in std_logic;
   sync         :  in std_logic;

   nrst         :  in std_logic;

   ccr_op       :  in std_logic_vector( 3 downto  0);

   alu_cond_sel :  in std_logic_vector( 1 downto  0);

   usq_cond_sel :  in std_logic_vector( 3 downto  0);

   alu_flags    :  in std_logic_vector( 5 downto  0);

   data_wr_db      :  in std_logic_vector( 7 downto  0);
   --
   --------------------------------------->


   ---------------------------------------> Outputs
   --
   alu_cond     :  out std_logic;
   usq_cond     :  out std_logic;

   ccr_data     :  out std_logic_vector( 7 downto  0)
   --
   ---------------------------------------<

);

--=============================================================================--

end ccr;

architecture behavior of ccr is

--==============================> Signals <==================================--


---------------------------------------> Operation Select Constants
--
constant  oooooooo     :  std_logic_vector(3 downto 0)  := x"0";
constant  ooooooo0     :  std_logic_vector(3 downto 0)  := x"1";
constant  ooooooo1     :  std_logic_vector(3 downto 0)  := x"2";
constant  oooooooX     :  std_logic_vector(3 downto 0)  := x"3";
constant  oooooo0o     :  std_logic_vector(3 downto 0)  := x"4";
constant  oooooo1o     :  std_logic_vector(3 downto 0)  := x"5";
constant  oooooXoo     :  std_logic_vector(3 downto 0)  := x"6";
constant  oooooXXX     :  std_logic_vector(3 downto 0)  := x"7";
constant  ooooXXXX     :  std_logic_vector(3 downto 0)  := x"8";
constant  ooooXXXo     :  std_logic_vector(3 downto 0)  := x"9";
constant  ooo0oooo     :  std_logic_vector(3 downto 0)  := x"A";
constant  ooo1oooo     :  std_logic_vector(3 downto 0)  := x"B";
constant  ooXoXXXX     :  std_logic_vector(3 downto 0)  := x"C";
constant  o1oooooo     :  std_logic_vector(3 downto 0)  := x"D";
constant  XVXXXXXX     :  std_logic_vector(3 downto 0)  := x"E";
--
----------------------------------------<

----------------------------------------> Internal CCR Signals
--
signal    alu_c        :  std_logic;   -- Carry
signal    alu_v        :  std_logic;   -- Overflow
signal    alu_z        :  std_logic;   -- Zero
signal    alu_n        :  std_logic;   -- Negative
signal    alu_h        :  std_logic;   -- Half Carry
signal    alu_a_sign       :  std_logic;   -- A Input Sign Bit

signal    c_reg        :  std_logic;   -- Carry
signal    v_reg        :  std_logic;   -- Overflow
signal    z_reg        :  std_logic;   -- Zero
signal    n_reg        :  std_logic;   -- Negative
signal    i_reg        :  std_logic;   -- I Interrupt Mask
signal    h_reg        :  std_logic;   -- Half Carry
signal    x_reg        :  std_logic;   -- X Interrupt Mask
signal    s_reg        :  std_logic;   -- Stop Disable

signal    bls_sig          :  std_logic;
signal    ble_sig          :  std_logic;
signal    blt_sig          :  std_logic;
--
----------------------------------------<

--==========================================================================--

begin

--=========================> CCR Architecture <=============================--

----------------------------------------> ALU Flag Input
--
alu_c     <= alu_flags( 0);
alu_v     <= alu_flags( 1);
alu_z     <= alu_flags( 2);
alu_n     <= alu_flags( 3);
alu_h     <= alu_flags( 4);
alu_a_sign   <= alu_flags( 5);
--
----------------------------------------<

----------------------------------------> Condition Code Register
--
process
(
   clk,
   sync,
   nrst,
   ccr_op,
   alu_c,
   alu_v,
   alu_z,
   alu_n,
   alu_h,
   data_wr_db
)
begin

   if (clk'event and clk = '1' and sync = '1') then

      if (nrst = '0')  then

         c_reg  <= '0';
         v_reg  <= '0';
         z_reg  <= '0';
         n_reg  <= '0';
         i_reg  <= '0';
         h_reg  <= '0';
         x_reg  <= '0';
         s_reg  <= '0';

      elsif (ccr_op = oooooooo) then

      -- c_reg  <= alu_c;
      -- v_reg  <= alu_v;
      -- z_reg  <= alu_z;
      -- n_reg  <= alu_n;
      -- i_reg  <= '0';
      -- h_reg  <= alu_h;
      -- x_reg  <= '0';
      -- s_reg  <= '0';

      elsif (ccr_op = ooooooo0) then

         c_reg  <= '0';
      -- v_reg  <= alu_v;
      -- z_reg  <= alu_z;
      -- n_reg  <= alu_n;
      -- i_reg  <= '0';
      -- h_reg  <= alu_h;
      -- x_reg  <= '0';
      -- s_reg  <= '0';

      elsif (ccr_op = ooooooo1) then

         c_reg  <= '1';
      -- v_reg  <= alu_v;
      -- z_reg  <= alu_z;
      -- n_reg  <= alu_n;
      -- i_reg  <= '0';
      -- h_reg  <= alu_h;
      -- x_reg  <= '0';
      -- s_reg  <= '0';

      elsif (ccr_op = oooooooX) then

         c_reg  <= alu_c;
      -- v_reg  <= alu_v;
      -- z_reg  <= alu_z;
      -- n_reg  <= alu_n;
      -- i_reg  <= '0';
      -- h_reg  <= alu_h;
      -- x_reg  <= '0';
      -- s_reg  <= '0';

      elsif (ccr_op = oooooo0o) then

      -- c_reg  <= alu_c;
         v_reg  <= '0';
      -- z_reg  <= alu_z;
      -- n_reg  <= alu_n;
      -- i_reg  <= '0';
      -- h_reg  <= alu_h;
      -- x_reg  <= '0';
      -- s_reg  <= '0';

      elsif (ccr_op = oooooo1o) then

      -- c_reg  <= alu_c;
         v_reg  <= '1';
      -- z_reg  <= alu_z;
      -- n_reg  <= alu_n;
      -- i_reg  <= '0';
      -- h_reg  <= alu_h;
      -- x_reg  <= '0';
      -- s_reg  <= '0';

      elsif (ccr_op = oooooXoo) then
```

```
-- c_reg  <= alu_c;
-- v_reg  <= alu_v;
   z_reg  <= alu_z;
-- n_reg  <= alu_n;
-- i_reg  <= '0';
-- h_reg  <= alu_h;
-- x_reg  <= '0';
-- s_reg  <= '0';

elsif (ccr_op = oooooXXX) then

   c_reg  <= alu_c;
   v_reg  <= alu_v;
   z_reg  <= alu_z;
-- n_reg  <= alu_n;
-- i_reg  <= '0';
-- h_reg  <= alu_h;
-- x_reg  <= '0';
-- s_reg  <= '0';

elsif (ccr_op = ooooXXXX) then

   c_reg  <= alu_c;
   v_reg  <= alu_v;
   z_reg  <= alu_z;
   n_reg  <= alu_n;
-- i_reg  <= '0';
-- h_reg  <= alu_h;
-- x_reg  <= '0';
-- s_reg  <= '0';

elsif (ccr_op = ooooXXXo) then

-- c_reg  <= alu_c;
   v_reg  <= alu_v;
   z_reg  <= alu_z;
   n_reg  <= alu_n;
-- i_reg  <= '0';
-- h_reg  <= alu_h;
-- x_reg  <= '0';
-- s_reg  <= '0';

elsif (ccr_op = ooo0oooo) then

-- c_reg  <= alu_c;
-- v_reg  <= alu_v;
-- z_reg  <= alu_z;
-- n_reg  <= alu_n;
   i_reg  <= '0';
-- h_reg  <= alu_h;
-- x_reg  <= '0';
-- s_reg  <= '0';

elsif (ccr_op = ooo1oooo) then

-- c_reg  <= alu_c;
-- v_reg  <= alu_v;
-- z_reg  <= alu_z;
-- n_reg  <= alu_n;
   i_reg  <= '1';
-- h_reg  <= alu_h;
-- x_reg  <= '0';
-- s_reg  <= '0';

elsif (ccr_op = ooXoXXXX) then

   c_reg  <= alu_c;
   v_reg  <= alu_v;
   z_reg  <= alu_z;
   n_reg  <= alu_n;
-- i_reg  <= '0';
   h_reg  <= alu_h;
```

```
        -- x_reg  <= '0';
        -- s_reg  <= '0';

        elsif (ccr_op = o1oooooo) then

        -- c_reg  <= alu_c;
        -- v_reg  <= alu_v;
        -- z_reg  <= alu_z;
        -- n_reg  <= alu_n;
        -- i_reg  <= '0';
        -- h_reg  <= alu_h;
           x_reg  <= '1';
        -- s_reg  <= '0';

        elsif (ccr_op = XVXXXXXX) then

           c_reg  <= data_wr_db(0);
           v_reg  <= data_wr_db(1);
           z_reg  <= data_wr_db(2);
           n_reg  <= data_wr_db(3);
           i_reg  <= data_wr_db(4);
           h_reg  <= data_wr_db(5);
           x_reg  <= data_wr_db(6) and x_reg;
           s_reg  <= data_wr_db(7);

        end if;

    end if;

end process;
--
----------------------------------------<

ccr_data(0)  <= c_reg;
ccr_data(1)  <= v_reg;
ccr_data(2)  <= z_reg;
ccr_data(3)  <= n_reg;
ccr_data(4)  <= i_reg;
ccr_data(5)  <= h_reg;
ccr_data(6)  <= x_reg;
ccr_data(7)  <= s_reg;


----------------------------------------> ALU Condition Select Logic
--
with alu_cond_sel select
alu_cond  <= '0'       when "00",    -- ZERO
             '1'       when "01",    -- ONE
             c_reg     when "10",    -- CCR_C
             alu_a_sign   when others;-- ASHIFT
--
----------------------------------------<

---------------------------------------->
--
ble_sig      <= z_reg or (n_reg xor v_reg);
blt_sig      <= n_reg xor v_reg;
bls_sig      <= c_reg or z_reg;
--
----------------------------------------<

----------------------------------------> Microsquencer Condition Select Logic
--
with usq_cond_sel select
usq_cond  <= '0'       when x"0",-- ZERO
             '1'       when x"1",-- ONE
             c_reg     when x"2",-- CCR_C
             v_reg     when x"3",-- CCR_V
             z_reg     when x"4",-- CCR_Z
             n_reg     when x"5",-- CCR_N
             ble_sig      when x"6",-- CCR_BLE
             blt_sig      when x"7",-- CCR_BLT
             bls_sig       when x"8",-- CCR_BLS
             '0'       when others;
--
----------------------------------------<

--==========================================================================--
end behavior;
```

```
--=========================> Gator uProccessor <===========================--
--
-- Engineer: Kevin Phillipson
-- Description:
--
--=========================================================================--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity mapper is port
(
   opcode     :  in  std_logic_vector( 7 downto  0);

   map0_vector  :  out std_logic_vector( 7 downto  0);
   map1_vector  :  out std_logic_vector( 7 downto  0);
   map2_vector  :  out std_logic_vector( 7 downto  0);
   map3_vector  :  out std_logic_vector( 7 downto  0);
   map4_vector  :  out std_logic_vector( 7 downto  0);
   map5_vector  :  out std_logic_vector( 7 downto  0)

);
end mapper;

architecture behavior of mapper is

begin

   with opcode select
   map0_vector  <=
      x"1D"  when   x"00",
      x"1E"  when   x"01",
      x"1F"  when   x"02",
      x"20"  when   x"03",
      x"21"  when   x"04",
      x"22"  when   x"05",
      x"23"  when   x"06",
      x"24"  when   x"07",
      x"25"  when   x"08",
      x"26"  when   x"09",
      x"27"  when   x"0A",
      x"28"  when   x"0B",
      x"29"  when   x"0C",
      x"2A"  when   x"0D",
      x"2B"  when   x"0E",
      x"2C"  when   x"0F",
      x"2D"  when   x"10",
      x"2E"  when   x"11",
      x"05"  when   x"12",
      x"05"  when   x"13",
      x"05"  when   x"14",
      x"05"  when   x"15",
      x"2F"  when   x"16",
      x"30"  when   x"17",
      x"1A"  when   x"18",
      x"31"  when   x"19",
      x"1B"  when   x"1A",
      x"32"  when   x"1B",
      x"11"  when   x"1C",
      x"11"  when   x"1D",
      x"11"  when   x"1E",
      x"11"  when   x"1F",
      x"19"  when   x"20",
      x"19"  when   x"21",
      x"19"  when   x"22",
      x"19"  when   x"23",
      x"19"  when   x"24",
      x"19"  when   x"25",
      x"19"  when   x"26",
      x"19"  when   x"27",
x"19"  when   x"28",
x"19"  when   x"29",
x"19"  when   x"2A",
x"19"  when   x"2B",
x"19"  when   x"2C",
x"19"  when   x"2D",
x"19"  when   x"2E",
x"19"  when   x"2F",
x"33"  when   x"30",
x"34"  when   x"31",
x"35"  when   x"32",
x"37"  when   x"33",
x"39"  when   x"34",
x"3A"  when   x"35",
x"3B"  when   x"36",
x"3C"  when   x"37",
x"3D"  when   x"38",
x"3F"  when   x"39",
x"41"  when   x"3A",
x"42"  when   x"3B",
x"43"  when   x"3C",
x"45"  when   x"3D",
x"46"  when   x"3E",
x"47"  when   x"3F",
x"48"  when   x"40",
x"49"  when   x"43",
x"4B"  when   x"44",
x"4C"  when   x"46",
x"4D"  when   x"47",
x"4E"  when   x"48",
x"4F"  when   x"49",
x"50"  when   x"4A",
x"51"  when   x"4C",
x"52"  when   x"4D",
x"53"  when   x"4F",
x"54"  when   x"50",
x"55"  when   x"53",
x"57"  when   x"54",
x"58"  when   x"56",
x"59"  when   x"57",
x"5A"  when   x"58",
x"5B"  when   x"59",
x"5C"  when   x"5A",
x"5D"  when   x"5C",
x"5E"  when   x"5D",
x"5F"  when   x"5F",
x"11"  when   x"60",
x"11"  when   x"63",
x"11"  when   x"64",
x"11"  when   x"66",
x"11"  when   x"67",
x"11"  when   x"68",
x"11"  when   x"69",
x"11"  when   x"6A",
x"11"  when   x"6C",
x"11"  when   x"6D",
x"17"  when   x"6E",
x"17"  when   x"6F",
x"0B"  when   x"70",
x"0B"  when   x"73",
x"0B"  when   x"74",
x"0B"  when   x"76",
x"0B"  when   x"77",
x"0B"  when   x"78",
x"0B"  when   x"79",
x"0B"  when   x"7A",
x"0B"  when   x"7C",
x"0B"  when   x"7D",
x"0F"  when   x"7E",
x"0F"  when   x"7F",
x"03"  when   x"80",
x"03"  when   x"81",
x"03"  when   x"82",
x"04"  when   x"83",
x"03"  when   x"84",
x"03"  when   x"85",
x"03"  when   x"86",
x"03"  when   x"88",
x"03"  when   x"89",
x"03"  when   x"8A",
x"03"  when   x"8B",
x"04"  when   x"8C",
x"19"  when   x"8D",
x"04"  when   x"8E",
x"60"  when   x"8F",
x"05"  when   x"90",
x"05"  when   x"91",
x"05"  when   x"92",
x"07"  when   x"93",
x"05"  when   x"94",
x"05"  when   x"95",
x"05"  when   x"96",
x"09"  when   x"97",
x"05"  when   x"98",
x"05"  when   x"99",
x"05"  when   x"9A",
x"05"  when   x"9B",
x"07"  when   x"9C",
x"09"  when   x"9D",
x"07"  when   x"9E",
x"09"  when   x"9F",
x"11"  when   x"A0",
x"11"  when   x"A1",
x"11"  when   x"A2",
x"14"  when   x"A3",
x"11"  when   x"A4",
x"11"  when   x"A5",
x"11"  when   x"A6",
x"17"  when   x"A7",
x"11"  when   x"A8",
x"11"  when   x"A9",
x"11"  when   x"AA",
x"11"  when   x"AB",
x"14"  when   x"AC",
x"17"  when   x"AD",
x"14"  when   x"AE",
x"17"  when   x"AF",
x"0B"  when   x"B0",
x"0B"  when   x"B1",
x"0B"  when   x"B2",
x"0D"  when   x"B3",
x"0B"  when   x"B4",
x"0B"  when   x"B5",
x"0B"  when   x"B6",
x"0F"  when   x"B7",
x"0B"  when   x"B8",
x"0B"  when   x"B9",
x"0B"  when   x"BA",
x"0B"  when   x"BB",
x"0D"  when   x"BC",
x"0F"  when   x"BD",
x"0D"  when   x"BE",
x"0F"  when   x"BF",
x"03"  when   x"C0",
x"03"  when   x"C1",
x"03"  when   x"C2",
x"04"  when   x"C3",
x"03"  when   x"C4",
x"03"  when   x"C5",
x"03"  when   x"C6",
x"03"  when   x"C8",
x"03"  when   x"C9",
x"03"  when   x"CA",
x"03"  when   x"CB",
        x"04"  when   x"CC",
        x"1C"  when   x"CD",
        x"04"  when   x"CE",
        x"63"  when   x"CF",
        x"05"  when   x"D0",
        x"05"  when   x"D1",
        x"05"  when   x"D2",
        x"07"  when   x"D3",
        x"05"  when   x"D4",
        x"05"  when   x"D5",
        x"05"  when   x"D6",
        x"09"  when   x"D7",
        x"05"  when   x"D8",
        x"05"  when   x"D9",
        x"05"  when   x"DA",
        x"05"  when   x"DB",
        x"07"  when   x"DC",
        x"09"  when   x"DD",
        x"07"  when   x"DE",
        x"09"  when   x"DF",
        x"11"  when   x"E0",
        x"11"  when   x"E1",
        x"11"  when   x"E2",
        x"14"  when   x"E3",
        x"11"  when   x"E4",
        x"11"  when   x"E5",
        x"11"  when   x"E6",
        x"17"  when   x"E7",
        x"11"  when   x"E8",
        x"11"  when   x"E9",
        x"11"  when   x"EA",
        x"11"  when   x"EB",
        x"14"  when   x"EC",
        x"17"  when   x"ED",
        x"14"  when   x"EE",
        x"17"  when   x"EF",
        x"0B"  when   x"F0",
        x"0B"  when   x"F1",
        x"0B"  when   x"F2",
        x"0D"  when   x"F3",
        x"0B"  when   x"F4",
        x"0B"  when   x"F5",
        x"0B"  when   x"F6",
        x"0F"  when   x"F7",
        x"0B"  when   x"F8",
        x"0B"  when   x"F9",
        x"0B"  when   x"FA",
        x"0B"  when   x"FB",
        x"0D"  when   x"FC",
        x"0F"  when   x"FD",
        x"0D"  when   x"FE",
        x"0F"  when   x"FF",
        x"FF"  when   others;

    with opcode select
    map1_vector  <=
        x"FF"  when   x"00",
        x"85"  when   x"12",
        x"86"  when   x"13",
        x"87"  when   x"14",
        x"88"  when   x"15",
        x"87"  when   x"1C",
        x"88"  when   x"1D",
        x"85"  when   x"1E",
        x"86"  when   x"1F",
        x"64"  when   x"20",
        x"66"  when   x"21",
        x"68"  when   x"22",
        x"6A"  when   x"23",
        x"6C"  when   x"24",
        x"6E"  when   x"25",
        x"70"  when   x"26",
x"72"  when   x"27",
x"74"  when   x"28",
x"76"  when   x"29",
x"78"  when   x"2A",
x"7A"  when   x"2B",
x"7C"  when   x"2C",
x"7E"  when   x"2D",
x"80"  when   x"2E",
x"82"  when   x"2F",
x"89"  when   x"60",
x"8A"  when   x"63",
x"8C"  when   x"64",
x"8D"  when   x"66",
x"8E"  when   x"67",
x"8F"  when   x"68",
x"90"  when   x"69",
x"91"  when   x"6A",
x"92"  when   x"6C",
x"93"  when   x"6D",
x"94"  when   x"6E",
x"95"  when   x"6F",
x"89"  when   x"70",
x"8A"  when   x"73",
x"8C"  when   x"74",
x"8D"  when   x"76",
x"8E"  when   x"77",
x"8F"  when   x"78",
x"90"  when   x"79",
x"91"  when   x"7A",
x"92"  when   x"7C",
x"93"  when   x"7D",
x"94"  when   x"7E",
x"95"  when   x"7F",
x"96"  when   x"80",
x"97"  when   x"81",
x"98"  when   x"82",
x"99"  when   x"83",
x"9A"  when   x"84",
x"9B"  when   x"85",
x"9C"  when   x"86",
x"9E"  when   x"88",
x"9F"  when   x"89",
x"A0"  when   x"8A",
x"A1"  when   x"8B",
x"A2"  when   x"8C",
x"84"  when   x"8D",
x"A5"  when   x"8E",
x"96"  when   x"90",
x"97"  when   x"91",
x"98"  when   x"92",
x"99"  when   x"93",
x"9A"  when   x"94",
x"9B"  when   x"95",
x"9C"  when   x"96",
x"9D"  when   x"97",
x"9E"  when   x"98",
x"9F"  when   x"99",
x"A0"  when   x"9A",
x"A1"  when   x"9B",
x"A2"  when   x"9C",
x"A3"  when   x"9D",
x"A5"  when   x"9E",
x"A6"  when   x"9F",
x"96"  when   x"A0",
x"97"  when   x"A1",
x"98"  when   x"A2",
x"99"  when   x"A3",
x"9A"  when   x"A4",
x"9B"  when   x"A5",
x"9C"  when   x"A6",
x"9D"  when   x"A7",
x"9E"  when   x"A8",
x"9F"  when   x"A9",
x"A0"  when   x"AA",
x"A1"  when   x"AB",
x"A2"  when   x"AC",
x"A3"  when   x"AD",
x"A5"  when   x"AE",
x"A6"  when   x"AF",
x"96"  when   x"B0",
x"97"  when   x"B1",
x"98"  when   x"B2",
x"99"  when   x"B3",
x"9A"  when   x"B4",
x"9B"  when   x"B5",
x"9C"  when   x"B6",
x"9D"  when   x"B7",
x"9E"  when   x"B8",
x"9F"  when   x"B9",
x"A0"  when   x"BA",
x"A1"  when   x"BB",
x"A2"  when   x"BC",
x"A3"  when   x"BD",
x"A5"  when   x"BE",
x"A6"  when   x"BF",
x"A7"  when   x"C0",
x"A8"  when   x"C1",
x"A9"  when   x"C2",
x"AA"  when   x"C3",
x"AB"  when   x"C4",
x"AC"  when   x"C5",
x"AD"  when   x"C6",
x"AF"  when   x"C8",
x"B0"  when   x"C9",
x"B1"  when   x"CA",
x"B2"  when   x"CB",
x"B3"  when   x"CC",
x"B5"  when   x"CE",
x"A7"  when   x"D0",
x"A8"  when   x"D1",
x"A9"  when   x"D2",
x"AA"  when   x"D3",
x"AB"  when   x"D4",
x"AC"  when   x"D5",
x"AD"  when   x"D6",
x"AE"  when   x"D7",
x"AF"  when   x"D8",
x"B0"  when   x"D9",
x"B1"  when   x"DA",
x"B2"  when   x"DB",
x"B3"  when   x"DC",
x"B4"  when   x"DD",
x"B5"  when   x"DE",
x"B6"  when   x"DF",
x"A7"  when   x"E0",
x"A8"  when   x"E1",
x"A9"  when   x"E2",
x"AA"  when   x"E3",
x"AB"  when   x"E4",
x"AC"  when   x"E5",
x"AD"  when   x"E6",
x"AE"  when   x"E7",
x"AF"  when   x"E8",
x"B0"  when   x"E9",
x"B1"  when   x"EA",
x"B2"  when   x"EB",
x"B3"  when   x"EC",
x"B4"  when   x"ED",
x"B5"  when   x"EE",
x"B6"  when   x"EF",
x"A7"  when   x"F0",
x"A8"  when   x"F1",
x"A9"  when   x"F2",
x"AA"  when   x"F3",
      x"AB"  when   x"F4",
      x"AC"  when   x"F5",
      x"AD"  when   x"F6",
      x"AE"  when   x"F7",
      x"AF"  when   x"F8",
      x"B0"  when   x"F9",
      x"B1"  when   x"FA",
      x"B2"  when   x"FB",
      x"B3"  when   x"FC",
      x"B4"  when   x"FD",
      x"B5"  when   x"FE",
      x"B6"  when   x"FF",
      x"FF"  when   others;

   with opcode select
   map2_vector  <=
      x"FF"  when   x"00",
      x"FF"  when   others;

   with opcode select
   map3_vector  <=
      x"FF"  when   x"00",
      x"FF"  when   others;

   with opcode select
   map4_vector  <=
      x"FF"  when   x"00",
      x"FF"  when   others;

   with opcode select
   map5_vector  <=
      x"FF"  when   x"00",
      x"FF"  when   others;

end behavior;
```

```
--==========================> Gator uProccessor <===========================--
-- MICROSQUENCER.VHD
-- Engineer: Kevin Phillipson
-- Date:     03/03/2007                              Revision:03.04.07
-- Description:
--
--=============================================================================--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity microsquencer is port
(
   nrst         :  in    std_logic;
   clk          :  in    std_logic;
   sync         :  in    std_logic;

   condition    :  in    std_logic;
   true_false      :  in    std_logic;

   micro_op     :  in    std_logic_vector( 2 downto  0);

   branch_vector:  in    std_logic_vector( 7 downto  0);

   map0_vector     :  in    std_logic_vector( 7 downto  0);
   map1_vector     :  in    std_logic_vector( 7 downto  0);
   map2_vector     :  in    std_logic_vector( 7 downto  0);
   map3_vector     :  in    std_logic_vector( 7 downto  0);
   map4_vector     :  in    std_logic_vector( 7 downto  0);
   map5_vector     :  in    std_logic_vector( 7 downto  0);

   micro_prog_addr  :  out   std_logic_vector( 7 downto  0)
);
end microsquencer;

architecture behavior of microsquencer is

----------------------------------------------------------------------->
Microprogram State Defines

constant  reset_vector :  std_logic_vector( 7 downto  0)   := (others => '0');
signal    state_reg    :  std_logic_vector( 7 downto  0);
signal    state_nxt    :  std_logic_vector( 7 downto  0);
signal    state_inc    :  std_logic_vector( 7 downto  0);

-----------------------------------------------------------------------<

constant  CONTINUE     :  std_logic_vector( 2 downto  0)   := "000";
constant  JUMP         :  std_logic_vector( 2 downto  0)   := "001";
constant  JUMP_MAP0    :  std_logic_vector( 2 downto  0)   := "010";
constant  JUMP_MAP1    :  std_logic_vector( 2 downto  0)   := "011";
constant  JUMP_MAP2    :  std_logic_vector( 2 downto  0)   := "100";
constant  JUMP_MAP3    :  std_logic_vector( 2 downto  0)   := "101";
constant  JUMP_MAP4    :  std_logic_vector( 2 downto  0)   := "110";
constant  JUMP_MAP5    :  std_logic_vector( 2 downto  0)   := "111";

begin

-----------------------------------------------------------------------> Next
State Logic
state_inc <= state_reg + '1';

process
(
   micro_op,
   condition,
   true_false,
   state_inc,
   branch_vector,
   map0_vector,
    map1_vector,
    map2_vector,
    map3_vector,
    map4_vector,
    map5_vector,
    state_reg
)
begin

    if (condition = true_false)   then

        case micro_op is

            when CONTINUE =>
                state_nxt <= state_inc;
            when JUMP =>
                state_nxt <= branch_vector;
            when JUMP_MAP0 =>
                state_nxt <= map0_vector;
            when JUMP_MAP1 =>
                state_nxt <= map1_vector;
            when JUMP_MAP2 =>
                state_nxt <= map2_vector;
            when JUMP_MAP3 =>
                state_nxt <= map3_vector;
            when JUMP_MAP4 =>
                state_nxt <= map4_vector;
            when JUMP_MAP5 =>
                state_nxt <= map5_vector;

        end case;

    else

        state_nxt <= state_inc;

    end if;

end process;
-----------------------------------------------------------------------<


----------------------------------------------------------------------->
Microprogram State Register
process
(
    nrst,
    clk,
    sync,
    state_nxt
)
begin
    if (clk'event and clk = '1') then

        if (nrst = '0') then

            state_reg <= reset_vector;

        elsif (sync = '1') then

            state_reg <= state_nxt;

        end if;

    end if;

end process;

micro_prog_addr  <= state_nxt when (sync = '1') else state_reg;
-----------------------------------------------------------------------<

end behavior;
```

```
--==========================> Gator uProccessor <===========================--
-- VECTOR_SPLIT.VHD
-- Engineer: Kevin Phillipson
-- Date:     02/09/2007                              Revision:4.01.07
-- Description:
--
--===========================================================================--

library ieee;
use ieee.std_logic_1164.all;

entity vector_split is

--============================> Port Map <=================================--

port
(

    ----------------------------------------> Microprogram Word
    --
    micro_prog_data  :  in std_logic_vector(55 downto  0);
    --
    ----------------------------------------<

    ----------------------------------------> Microsquencer
    --
    true_false       :  out std_logic;
    micro_op      :  out std_logic_vector( 2 downto  0);
    branch_addr      :  out std_logic_vector( 7 downto  0);
    --
    ----------------------------------------<

    ----------------------------------------> CCR
    --
    ccr_op        :  out std_logic_vector( 4 downto  0);
    alu_cond_sel :  out std_logic_vector( 1 downto  0);
    usq_cond_sel :  out std_logic_vector( 3 downto  0);
    --
    ----------------------------------------<

    ----------------------------------------> Register Array
    --
    addr_sel      :  out std_logic_vector( 3 downto  0);
    data_a_sel       :  out std_logic_vector( 3 downto  0);
    data_b_sel       :  out std_logic_vector( 3 downto  0);
    data_wr_sel      :  out std_logic_vector( 3 downto  0);
    --
    ----------------------------------------<

    ----------------------------------------> Adress ALU
    --
    addr_alu_op      :  out std_logic_vector( 3 downto  0);
    --
    ----------------------------------------<

    ----------------------------------------> Data ALU
    --
    data_alu_op      :  out std_logic_vector( 2 downto  0);
    data_alu_mode :  out std_logic;
    --
    ----------------------------------------<

    ----------------------------------------> Memory Controller
    --
    mem_func_sel :  out std_logic_vector( 2 downto  0)
    --
    ----------------------------------------<

);

--===========================================================================--
```

```
end vector_split;

architecture behavior of vector_split is

--==============================> Signals <==================================--

--=========================================================================--

begin

   ----------------------------------------> Microsquencer
   --
   micro_op      <= micro_prog_data(55 downto 53);   -- 3
   true_false       <= micro_prog_data(52);          -- 1
   branch_addr      <= micro_prog_data(51 downto 44);   -- 8
   --
   ----------------------------------------<

   ----------------------------------------> CCR
   --
   ccr_op        <= micro_prog_data(43 downto 39);   -- 5
   alu_cond_sel <= micro_prog_data(38 downto 37);  -- 2
   usq_cond_sel <= micro_prog_data(36 downto 33);   -- 4
   --
   ----------------------------------------<

   ----------------------------------------> Register Array
   --
   addr_sel      <= micro_prog_data(32 downto 29);   -- 4
   data_a_sel       <= micro_prog_data(28 downto 25);   -- 4
   data_b_sel       <= micro_prog_data(24 downto 21);   -- 4
   data_wr_sel      <= micro_prog_data(20 downto 17);   -- 4
   --
   ----------------------------------------<

   ----------------------------------------> Adress ALU
   --
   addr_alu_op      <= micro_prog_data(16 downto 13);   -- 4
   --
   ----------------------------------------<

   ----------------------------------------> Data ALU
   --
   data_alu_op      <= micro_prog_data(12 downto 10);   -- 3
   data_alu_mode<= micro_prog_data( 9);          -- 1
   --
   ----------------------------------------<

   ----------------------------------------> Memory Controller
   --
   mem_func_sel <= micro_prog_data( 8 downto  6);   -- 3
   --
   ----------------------------------------<


end behavior;
```

```
; --==========================> Gator uProccessor <===========================-
-
; -- File:          GUCODE.ASM
; -- Engineer:      Kevin Phillipson
; -- Credit:    Based on concepts from
;         MICROPROGRAMMED STATE MACHINE DESIGN
;         by Dr. Michel M. Lynch
; -- Date:      03.16.07
; -- Revision:      03.16.07
; -- Description:
; --=========================================================================-
-


   nolist

   include   "gucode.mac"

   list

   ORGA   $00

RESET:

   DATA16_PASS   ZERO
   DATA_WRITE    PC

   CCR_OP    o1oooooo
   end_state

FETCH:

   LOAD_OPCODE
   ADDR_POST_INC PC
   end_state

DECODE:

   JUMP_MAP0
   end_state

; --================================> PAGE 0 <================================-
-

; ---------------------------------> MAP 0

; ADDRESSING MODES

LOAD_IMM8:
   set_map0  80 81 82    ;SUBA CMPA SBCA
   set_map0  84 85 86    ;ANDA BITA LDAA
   set_map0  88 89 8A 8B ;EORA ADCA ORAA ADDA

   set_map0  C0 C1 C2    ;SUBB CMPB SBCB
   set_map0  C4 C5 C6    ;ANDB BITB LDAB
   set_map0  C8 C9 CA CB ;EORB ADCB ORAB ADDB

   DATA16_PASS   PC
   DATA_WRITE    EA

   ADDR_POST_INC PC
   LOAD_DATA8

   JUMP_MAP1
   end_state

LOAD_IMM16:
   set_map0  83          ;SUBD
   set_map0  8C 8E       ;CPX LDS

   set_map0  C3          ;ADDD
   set_map0  CC CE       ;LDD LDX
```

```
    DATA16_PASS  PC
    DATA_WRITE   EA

    ADDR_POST_INC2   PC
    LOAD_DATA16

    JUMP_MAP1
    end_state

LOAD_DIR8:
    set_map0  12 13 14 15 ;BSET BCLR BRSET BRCLR

    set_map0  90 91 92    ;SUBA CMPA SBCA SUBD
    set_map0  94 95 96    ;ANDA BITA LDAA STAA
    set_map0  98 99 9A 9B ;EORA ADCA ORAA ADDA
    set_map0              ;CPX JSR LDS STS

    set_map0  D0 D1 D2    ;SUBB CMPB SBCB ADDD
    set_map0  D4 D5 D6    ;ANDB BITB LDAB STAB
    set_map0  D8 D9 DA DB ;EORB ADCB ORAB ADDB
    set_map0              ;LDD STD LDX STX

    ADDR_POST_INC PC
    LOAD_DATA8
    end_state

    DATA16_PASS  MEM_U8
    DATA_WRITE   EA

    ADDR_PASS MEM_U8
    LOAD_DATA8

    JUMP_MAP1
    end_state

LOAD_DIR16:
    set_map0              ;BSET BCLR BRSET BRCLR

    set_map0           93 ;SUBA CMPA SBCA SUBD
    set_map0              ;ANDA BITA LDAA STAA
    set_map0              ;EORA ADCA ORAA ADDA
    set_map0  9C    9E    ;CPX JSR LDS STS

    set_map0           D3 ;SUBB CMPB SBCB ADDD
    set_map0              ;ANDB BITB LDAB STAB
    set_map0              ;EORB ADCB ORAB ADDB
    set_map0  DC    DE    ;LDD STD LDX STX

    ADDR_POST_INC PC
    LOAD_DATA8
    end_state

    DATA16_PASS  MEM_U8
    DATA_WRITE   EA

    ADDR_PASS MEM_U8
    LOAD_DATA16

    JUMP_MAP1
    end_state

STORE_DIR:
    set_map0              ;BSET BCLR BRSET BRCLR

    set_map0              ;SUBA CMPA SBCA SUBD
    set_map0           97 ;ANDA BITA LDAA STAA
    set_map0              ;EORA ADCA ORAA ADDA
    set_map0     9D    9F ;CPX JSR LDS STS

    set_map0              ;SUBB CMPB SBCB ADDD
    set_map0           D7 ;ANDB BITB LDAB STAB
    set_map0               ;EORB ADCB ORAB ADDB
    set_map0     DD    DF ;LDD STD LDX STX

    ADDR_POST_INC PC
    LOAD_DATA8
    end_state

    DATA16_PASS  MEM_U8
    DATA_WRITE   EA

    JUMP_MAP1
    end_state

LOAD_EXT8:
    set_map0  70 73 74 76 ;NEG COM LSR ROR
    set_map0  77 78 79 7A ;ASR ASL ROL DEC
    set_map0  7C 7D       ;INC TST JMP CLR

    set_map0  B0 B1 B2    ;SUBA CMPA SBCA SUBD
    set_map0  B4 B5 B6    ;ANDA BITA LDAA STAA
    set_map0  B8 B9 BA BB ;EORA ADCA ORAA ADDA
    set_map0              ;CPX JSR LDS STS

    set_map0  F0 F1 F2    ;SUBB CMPB SBCB ADDD
    set_map0  F4 F5 F6    ;ANDB BITB LDAB STAB
    set_map0  F8 F9 FA FB ;EORB ADCB ORAB ADDB
    set_map0              ;LDD STD LDX STX

    ADDR_POST_INC2   PC
    LOAD_DATA16
    end_state

    DATA16_PASS  MEM_U16
    DATA_WRITE   EA

    ADDR_PASS MEM_U16
    LOAD_DATA8

    JUMP_MAP1
    end_state

LOAD_EXT16:
    set_map0              ;NEG COM LSR ROR
    set_map0              ;ASR ASL ROL DEC
    set_map0              ;INC TST JMP CLR

    set_map0           B3 ;SUBA CMPA SBCA SUBD
    set_map0              ;ANDA BITA LDAA STAA
    set_map0              ;EORA ADCA ORAA ADDA
    set_map0  BC    BE    ;CPX JSR LDS STS

    set_map0           F3 ;SUBB CMPB SBCB ADDD
    set_map0              ;ANDB BITB LDAB STAB
    set_map0              ;EORB ADCB ORAB ADDB
    set_map0  FC    FE    ;LDD STD LDX STX

    ADDR_POST_INC2   PC
    LOAD_DATA16
    end_state

    DATA16_PASS  MEM_U16
    DATA_WRITE   EA

    ADDR_PASS MEM_U16
    LOAD_DATA16

    JUMP_MAP1
    end_state

STORE_EXT:
    set_map0              ;NEG COM LSR ROR
    set_map0              ;ASR ASL ROL DEC
    set_map0        7E 7F ;INC TST JMP CLR

    set_map0              ;SUBA CMPA SBCA SUBD
    set_map0           B7 ;ANDA BITA LDAA STAA
    set_map0              ;EORA ADCA ORAA ADDA
    set_map0     BD    BF ;CPX JSR LDS STS

    set_map0              ;SUBB CMPB SBCB ADDD
    set_map0           F7 ;ANDB BITB LDAB STAB
    set_map0              ;EORB ADCB ORAB ADDB
    set_map0     FD    FF ;LDD STD LDX STX

    ADDR_POST_INC2   PC
    LOAD_DATA16
    end_state

    DATA16_PASS  MEM_U16
    DATA_WRITE   EA

    JUMP_MAP1
    end_state

LOAD_IDX8:
    set_map0  1C 1D 1E 1F ;BSET BCLR BRSET BRCLR

    set_map0  60 63 64 66 ;NEG COM LSR ROR
    set_map0  67 68 69 6A ;ASR ASL ROL DEC
    set_map0  6C 6D       ;INC TST JMP CLR

    set_map0  A0 A1 A2    ;SUBA CMPA SBCA SUBD
    set_map0  A4 A5 A6    ;ANDA BITA LDAA STAA
    set_map0  A8 A9 AA AB ;EORA ADCA ORAA ADDA
    set_map0              ;CPX JSR LDS STS

    set_map0  E0 E1 E2    ;SUBB CMPB SBCB ADDD
    set_map0  E4 E5 E6    ;ANDB BITB LDAB STAB
    set_map0  E8 E9 EA EB ;EORB ADCB ORAB ADDB
    set_map0              ;LDD STD LDX STX

    ADDR_POST_INC PC
    LOAD_DATA8
    end_state

    DATA16_ADD   MEM_U8,X
    DATA_WRITE   EA
    end_state

    ADDR_PASS EA
    LOAD_DATA8

    JUMP_MAP1
    end_state

LOAD_IDX16:
    set_map0              ;BSET BCLR BRSET BRCLR

    set_map0              ;NEG COM LSR ROR
    set_map0              ;ASR ASL ROL DEC
    set_map0              ;INC TST JMP CLR

    set_map0           A3 ;SUBA CMPA SBCA SUBD
    set_map0              ;ANDA BITA LDAA STAA
    set_map0              ;EORA ADCA ORAA ADDA
    set_map0  AC    AE    ;CPX JSR LDS STS

    set_map0           E3 ;SUBB CMPB SBCB ADDD
    set_map0              ;ANDB BITB LDAB STAB
    set_map0              ;EORB ADCB ORAB ADDB
    set_map0  EC    EE    ;LDD STD LDX STX

    ADDR_POST_INC PC
    LOAD_DATA8
    end_state

    DATA16_ADD    MEM_U8,X
    DATA_WRITE    EA
    end_state

    ADDR_PASS EA
    LOAD_DATA16

    JUMP_MAP1
    end_state

STORE_IDX:
    set_map0              ;BSET BCLR BRSET BRCLR

    set_map0              ;NEG COM LSR ROR
    set_map0              ;ASR ASL ROL DEC
    set_map0        6E 6F ;INC TST JMP CLR

    set_map0              ;SUBA CMPA SBCA SUBD
    set_map0           A7 ;ANDA BITA LDAA STAA
    set_map0              ;EORA ADCA ORAA ADDA
    set_map0     AD    AF ;CPX JSR LDS STS

    set_map0              ;SUBB CMPB SBCB ADDD
    set_map0           E7 ;ANDB BITB LDAB STAB
    set_map0              ;EORB ADCB ORAB ADDB
    set_map0     ED    EF ;LDD STD LDX STX

    ADDR_POST_INC PC
    LOAD_DATA8
    end_state

    DATA16_ADD    MEM_U8,X
    DATA_WRITE    EA

    JUMP_MAP1
    end_state

REL:
    set_map0  20 21 22 23 ;BRA BRN BHI BLS
    set_map0  24 25 26 27 ;BCC BCS BNE BEQ
    set_map0  28 29 2A 2B ;BVC BVS BPL BMI
    set_map0  2C 2D 2E 2F ;BGE BLT BGT BLE
    set_map0  8D          ;BSR

    ADDR_POST_INC PC
    LOAD_DATA8

    JUMP_MAP1
    end_state

PAGE_2:
    set_map0  18

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP_MAP2
    end_state

PAGE_3:
    set_map0  1A

    JUMP      TRAP
    end_state

PAGE_4:
    set_map0  CD

    JUMP      TRAP
    end_state
```

```
; INHERENT INSTRUCTIONS

TEST:
   set_map0  00

   JUMP      TRAP
   end_state

NOP:
   set_map0  01

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

IDIV:            ;NOT YET IMPLEMENTED
   set_map0  02

   JUMP      TRAP
   end_state

FDIV:            ;NOT YET IMPLEMENTED
   set_map0  03

   JUMP      TRAP
   end_state

LSRD:
   set_map0  04

   DATA16_RSHIFT ZERO,D
   DATA_WRITE    D
   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

ASLD:
   set_map0  05

   DATA16_LSHIFT D,ZERO
   DATA_WRITE    D
   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

TAP:
   set_map0  06

   DATA8_PASS    A

   CCR_OP    XVXXXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

TPA:
   set_map0  07
```

```
        DATA8_PASS    CCR
        DATA_WRITE    A

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

INX:    set_map0  08

        DATA16_INC    X
        DATA_WRITE    X
        CCR_OP    oooooXoo

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

DEX:    set_map0  09

        DATA16_DEC    X
        DATA_WRITE    X
        CCR_OP    oooooXoo

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

CLV:    set_map0  0A

        CCR_OP    oooooo0o

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

SEV:    set_map0  0B

        CCR_OP    oooooo1o

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

CLC:    set_map0  0C

        CCR_OP    ooooooo0

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

SEC:    set_map0  0D

        CCR_OP    ooooooo1

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state
```

```
CLI:   set_map0  0E

    CCR_OP    ooo0oooo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

SEI:   set_map0  0F

    CCR_OP    ooo1oooo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

SBA:   set_map0  10

    DATA8_SUB A,B
    DATA_WRITE    A

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

CBA:   set_map0  11

    DATA8_SUB A,B

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

TAB:
    set_map0  16

    DATA8_PASS    A
    DATA_WRITE    B

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

TBA:
    set_map0  17

    DATA8_PASS    B
    DATA_WRITE    A

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state


DAA:            ;NOT YET IMPLEMENTED
    set_map0  19

    JUMP      TRAP
    end_state

ABA:  set_map0  1B

    DATA8_ADD A,B
    DATA_WRITE    A

    CCR_OP    ooXoXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

TSX:
    set_map0  30

    DATA16_INC    SP
    DATA_WRITE    X

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

INS:
    set_map0  31

    DATA16_INC    SP
    DATA_WRITE    SP

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

PULA:
    set_map0  32

    ADDR_PRE_INC  SP
    LOAD_DATA8
    end_state

    DATA8_PASS    MEM_U8
    DATA_WRITE    A

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

PULB:
    set_map0  33

    ADDR_PRE_INC  SP
    LOAD_DATA8
    end_state

    DATA8_PASS    MEM_U8
    DATA_WRITE    B
    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

DES:
    set_map0  34

    DATA16_DEC    SP
    DATA_WRITE    SP

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

TXS:
    set_map0  35

    DATA16_DEC    X
    DATA_WRITE    SP

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

PSHA:
    set_map0  36

    DATA8_PASS    A

    ADDR_POST_DEC SP
    STORE_DATA8

    JUMP      FETCH
    end_state

PSHB:
    set_map0  37

    DATA8_PASS    B

    ADDR_POST_DEC SP
    STORE_DATA8

    JUMP      FETCH
    end_state

PULX:
    set_map0  38

    ADDR_PRE_INC  SP
    LOAD_DATA16
    end_state

    DATA16_PASS   MEM_U16
    DATA_WRITE    X

    ADDR_PRE_INC  SP

    JUMP      FETCH
    end_state

RTS:
    set_map0  39

    ADDR_PRE_INC  SP
    LOAD_DATA16
    end_state

    DATA16_PASS  MEM_U16
    DATA_WRITE   PC

    ADDR_PRE_INC SP

    JUMP      FETCH
    end_state

ABX:
    set_map0  3A

    DATA16_ADD   X,B
    DATA_WRITE   X

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

RTI:             ;NOT YET IMPLEMENTED
    set_map0  3B

    JUMP      TRAP
    end_state

PSHX:
    set_map0  3C

    DATA16_PASS  X

    ADDR_PRE_DEC SP
    STORE_DATA16
    end_state

    ADDR_PRE_DEC SP

    JUMP      FETCH
    end_state

MUL:             ;NOT YET IMPLEMENTED
    set_map0  3D

    JUMP      TRAP
    end_state

WAI:             ;NOT YET IMPLEMENTED
    set_map0  3E

    JUMP      TRAP
    end_state

SWI:             ;NOT YET IMPLEMENTED
    set_map0  3F

    JUMP      TRAP
    end_state

NEGA:
    set_map0  40

    DATA8_SUB ZERO,A
    DATA_WRITE   A

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

COMA:  set_map0  43

    DATA8_NOT A
    DATA_WRITE   A

    CCR_OP    ooooXXXX
    end_state

    CCR_OP    ooooooo1

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

LSRA:  set_map0  44

    DATA8_RSHIFT ZERO,A
    DATA_WRITE   A
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

RORA:  set_map0  46

    DATA8_RSHIFT CCR_C,A
    DATA_WRITE   A
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ASRA:  set_map0  47

    DATA8_RSHIFT ASHIFT,A
    DATA_WRITE   A
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ASLA:  set_map0  48

    DATA8_LSHIFT A,ZERO
    DATA_WRITE   A
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ROLA:  set_map0  49

    DATA8_LSHIFT A,CCR_C
    DATA_WRITE   A
    CCR_OP    ooooXXXX
```

```
        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

DECA:   set_map0  4A

        DATA8_DEC A
        DATA_WRITE    A
        CCR_OP    ooooXXXo

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

INCA:   set_map0  4C

        DATA8_INC A
        DATA_WRITE    A
        CCR_OP    ooooXXXo

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

TSTA:   set_map0  4D

        DATA8_SUB A,ZERO
        DATA_WRITE    A
        CCR_OP    ooooXXXX

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

CLRA:   set_map0  4F

        DATA8_PASS    ZERO
        DATA_WRITE    A
        CCR_OP    ooooXXXX

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

NEGB:
        set_map0  50

        DATA8_SUB ZERO,B
        DATA_WRITE    B

        CCR_OP    ooooXXXX

        ADDR_POST_INC PC
        LOAD_OPCODE

        JUMP      DECODE
        end_state

COMB:   set_map0  53

        DATA8_NOT B
        DATA_WRITE    B
```

```
    CCR_OP    ooooXXXX
    end_state

    CCR_OP    ooooooo1

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

LSRB:  set_map0  54

    DATA8_RSHIFT ZERO,B
    DATA_WRITE   B
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

RORB:  set_map0  56

    DATA8_RSHIFT CCR_C,B
    DATA_WRITE   B
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ASRB:  set_map0  57

    DATA8_RSHIFT ASHIFT,B
    DATA_WRITE   B
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ASLB:  set_map0  58

    DATA8_LSHIFT B,ZERO
    DATA_WRITE   B
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ROLB:  set_map0  59

    DATA8_LSHIFT B,CCR_C
    DATA_WRITE   B
    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state
```

```
DECB:  set_map0  5A

   DATA8_DEC B
   DATA_WRITE   B
   CCR_OP    ooooXXXo

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

INCB:  set_map0  5C

   DATA8_INC B
   DATA_WRITE   B
   CCR_OP    ooooXXXo

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

TSTB:  set_map0  5D

   DATA8_SUB B,ZERO
   DATA_WRITE   B
   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

CLRB:
   set_map0  5F

   DATA8_PASS   ZERO
   DATA_WRITE   B
   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

XGDX:
   set_map0  8F

   DATA16_PASS  X
   DATA_WRITE   EA
   end_state

   DATA16_PASS  D
   DATA_WRITE   X
   end_state

   DATA16_PASS  EA
   DATA_WRITE   D

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

STOP:
   set_map0  CF
```

```
    JUMP      TRAP
    end_state


; ---------------------------------> MAP 1


BRA:
    set_map1  20

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    IF     ZERO,CLEAR
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BRN:
    set_map1  21

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    IF     ZERO,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BHI:
    set_map1  22

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    IF     BLS,CLEAR
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BLS:
    set_map1  23

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    IF     BLS,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BCC:
    set_map1  24

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_C,CLEAR
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BCS:
    set_map1  25

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_C,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BNE:
    set_map1  26

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_Z,CLEAR
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BEQ:
    set_map1  27

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_Z,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BVC:
    set_map1  28

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_V,CLEAR
    JUMP      JMP
    end_state
    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BVS:
    set_map1  29

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_V,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BPL:
    set_map1  2A

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_N,CLEAR
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BMI:
    set_map1  2B

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_N,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BGE:
    set_map1  2C

    DATA16_ADD    PC,MEM_S8
    DATA_WRITE    EA

    IF     CCR_BLT,CLEAR
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BLT:
    set_map1  2D

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    IF     CCR_BLT,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BGT:
    set_map1  2E

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    IF     CCR_BLE,CLEAR
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BLE:
    set_map1  2F

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    IF     CCR_BLE,SET
    JUMP      JMP
    end_state

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BSR:
    set_map1  8D

    DATA16_ADD   PC,MEM_S8
    DATA_WRITE   EA

    JUMP      JSR
    end_state


BRSET:           ;NOT YET IMPLEMENTED
    set_map1  12 1E

    JUMP      TRAP
    end_state

BRCLR:           ;NOT YET IMPLEMENTED
    set_map1  13 1F

    JUMP      TRAP
    end_state

BSET:            ;NOT YET IMPLEMENTED
    set_map1  14 1C
    JUMP      TRAP
    end_state

BCLR:           ;NOT YET IMPLEMENTED
    set_map1  15 1D

    JUMP      TRAP
    end_state


NEG:
    set_map1  60 70

    DATA8_SUB ZERO,MEM_U8

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

COM:   set_map1  63 73

    DATA8_NOT MEM_U8

    CCR_OP    ooooXXXX
    end_state

    CCR_OP    ooooooo1

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

LSR:   set_map1  64 74

    DATA8_RSHIFT ZERO,MEM_U8

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

ROR:   set_map1  66 76

    DATA8_RSHIFT CCR_C,MEM_U8

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

ASR:   set_map1  67 77

    DATA8_RSHIFT ASHIFT,MEM_U8

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

ASL:    set_map1  68 78

    DATA8_LSHIFT MEM_U8,ZERO

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

ROL:    set_map1  69 79

    DATA8_LSHIFT MEM_U8,CCR_C

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

DEC:    set_map1  6A 7A

    DATA8_DEC MEM_U8

    CCR_OP    ooooXXXo

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

INC:    set_map1  6C 7C

    DATA8_INC MEM_U8

    CCR_OP    ooooXXXo

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

TST:    set_map1  6D 7D

    DATA8_SUB MEM_U8,ZERO

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state


JMP:
    set_map1  6E 7E

    DATA16_INC   EA
    DATA_WRITE   PC

    ADDR_PASS EA
    LOAD_OPCODE
    JUMP      DECODE
    end_state



CLR:  set_map1  6F 7F

    DATA8_PASS    ZERO

    CCR_OP    ooooXXXX

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

;

SUBA:
    set_map1  80 90 A0 B0

    DATA8_SUB A,MEM_U8
    DATA_WRITE    A

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

CMPA:
    set_map1  81 91 A1 B1

    DATA8_SUB A,MEM_U8

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

SBCA:
    set_map1  82 92 A2 B2

    DATA8_SUBC    A,MEM_U8
    DATA_WRITE    A

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

SUBD:
    set_map1  83 93 A3 B3

    DATA16_SUB    D,MEM_U8
    DATA_WRITE    D

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE
    JUMP      DECODE
    end_state

ANDA:
    set_map1  84 94 A4 B4

    DATA8_AND A,MEM_U8
    DATA_WRITE   A

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

BITA:
    set_map1  85 95 A5 B5

    DATA8_AND A,MEM_U8

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

LDAA:
    set_map1  86 96 A6 B6

    DATA8_PASS   MEM_U8
    DATA_WRITE   A

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state


STAA:
    set_map1  97 A7 B7

    DATA8_PASS   A

    CCR_OP    ooooXXXo

    ADDR_PASS EA
    STORE_DATA8

    JUMP      FETCH
    end_state

EORA:
    set_map1  88 98 A8 B8

    DATA8_XOR A,MEM_U8
    DATA_WRITE   A

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state
```

```
ADCA:
   set_map1  89 99 A9 B9

   DATA8_ADDC   A,MEM_U8
   DATA_WRITE   A

   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

ORAA:
   set_map1  8A 9A AA BA

   DATA8_OR  A,MEM_U8
   DATA_WRITE   A

   CCR_OP    ooooXXXo

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

ADDA:
   set_map1  8B 9B AB BB

   DATA8_ADD A,MEM_U8
   DATA_WRITE   A

   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

CPX:
   set_map1  8C 9C AC BC

   DATA16_SUB   X,MEM_U16

   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

JSR:
   set_map1  9D AD BD

   DATA16_PASS  PC

   ADDR_PRE_DEC SP
   STORE_DATA16
   end_state

   DATA16_PASS  EA
   DATA_WRITE   PC

   ADDR_PRE_DEC SP

   JUMP      FETCH
   end_state
```

```
LDS:
   set_map1  8E 9E AE BE

   DATA16_PASS  MEM_U16
   DATA_WRITE   SP

   CCR_OP    ooooXXXo

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

STS:
   set_map1  9F AF BF

   DATA16_PASS  SP

   CCR_OP    ooooXXXo

   ADDR_PASS EA
   STORE_DATA16

   JUMP      FETCH
   end_state


;


SUBB:
   set_map1  C0 D0 E0 F0

   DATA8_SUB B,MEM_U8
   DATA_WRITE   B

   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

CMPB:
   set_map1  C1 D1 E1 F1

   DATA8_SUB B,MEM_U8

   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

SBCB:
   set_map1  C2 D2 E2 F2

   DATA8_SUBC   B,MEM_U8
   DATA_WRITE   B

   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state
```

```
ADDD:
   set_map1  C3 D3 E3 F3

   DATA16_ADD   D,MEM_U8
   DATA_WRITE   D

   CCR_OP    ooooXXXX

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

ANDB:
   set_map1  C4 D4 E4 F4

   DATA8_AND B,MEM_U8
   DATA_WRITE   B

   CCR_OP    ooooXXXo

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

BITB:
   set_map1  C5 D5 E5 F5

   DATA8_AND B,MEM_U8

   CCR_OP    ooooXXXo

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state

LDAB:
   set_map1  C6 D6 E6 F6

   DATA8_PASS   MEM_U8
   DATA_WRITE   B

   CCR_OP    ooooXXXo

   ADDR_POST_INC PC
   LOAD_OPCODE

   JUMP      DECODE
   end_state


STAB:
   set_map1  D7 E7 F7

   DATA8_PASS   B

   CCR_OP    ooooXXXo

   ADDR_PASS EA
   STORE_DATA8

   JUMP      FETCH
   end_state

EORB:
   set_map1  C8 D8 E8 F8
```

```
    DATA8_XOR B,MEM_U8
    DATA_WRITE   B

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ADCB:
    set_map1  C9 D9 E9 F9

    DATA8_ADDC   B,MEM_U8
    DATA_WRITE   B

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ORAB:
    set_map1  CA DA EA FA

    DATA8_OR  B,MEM_U8
    DATA_WRITE   B

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

ADDB:
    set_map1  CB DB EB FB

    DATA8_ADD B,MEM_U8
    DATA_WRITE   B

    CCR_OP    ooooXXXX

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state

LDD:
    set_map1  CC DC EC FC

    DATA16_PASS  MEM_U16
    DATA_WRITE   D

    CCR_OP    ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP      DECODE
    end_state


STD:
    set_map1  DD ED FD

    DATA16_PASS  D
    CCR_OP      ooooXXXo

    ADDR_PASS EA
    STORE_DATA16

    JUMP        FETCH
    end_state

LDX:
    set_map1  CE DE EE FE

    DATA16_PASS   MEM_U16
    DATA_WRITE    X

    CCR_OP      ooooXXXo

    ADDR_POST_INC PC
    LOAD_OPCODE

    JUMP        DECODE
    end_state

STX:
    set_map1  DF EF FF

    DATA16_PASS   X

    CCR_OP      ooooXXXo

    ADDR_PASS EA
    STORE_DATA16

    JUMP        FETCH
    end_state

; --=================================> PAGE 1 <=================================-
-

; ----------------------------------> MAP 2
; ADDRESSING MODES


    ORGA   $FF

TRAP:
    JUMP        TRAP
    end_state

    END
```

```
; --=========================> Gator uProccessor <==========================--
; -- File:          GUCODE.MAC
; -- Engineer:      Kevin Phillipson
; -- Credit:     Based on concepts from
;         MICROPROGRAMMED STATE MACHINE DESIGN
;         by Dr. Michel M. Lynch
; -- Date:      03.16.07
; -- Revision:      03.16.07
; -- Description:
; --========================================================================--

; Assembler operators used in expressions:
;  + add two symbols
;  - subtract two symbols
;  * multiply two symbols
;  / divide two symbols
;  ! logical OR
;  & logical AND
;  >>N shift operand N-bits right
;  <<N shift operand N-bits left

; Assembler Data Declaratives
; Label:  dc.b  operand   - initialize a byte of memory with operand
; Label:  dc.w  operand   - initialize a 16-bit word of memory with operand
; Label:  dc.l  operand   - initialize a 32-bit word of memory with operand
; Label:  dc.b  N   - set aside N bytes of memory
; Label:  dc.w  N   - set aside N 16-bit words of memory
; Label:  dc.l  N   - set aside N 32-bit words of memory


; --============================> Vector Defines <=========================--

WORD_WIDTH   EQU 16

; ----------------------------------------> micro_op vec defines
; --
CONTINUE_VEC EQU $0
JUMP_VEC  EQU $1
JUMP_MAP0_VEC EQU $2
JUMP_MAP1_VEC EQU $3
JUMP_MAP2_VEC EQU $4
JUMP_MAP3_VEC EQU $5
JUMP_MAP4_VEC EQU $6
JUMP_MAP5_VEC EQU $7
; --
; ----------------------------------------<

; ----------------------------------------> true_false vec defines
; --
FALSE     EQU $0
TRUE         EQU $1
CLEAR     EQU $0
SET    EQU $1
; --
; ----------------------------------------<

; ----------------------------------------> ccr_op vec defines
; --
oooooooo  EQU $0
ooooooo0  EQU $1
ooooooo1  EQU $2
oooooooX  EQU $3
oooooo0o  EQU $4
oooooo1o  EQU $5
oooooXoo  EQU $6
oooooXXX  EQU $7
ooooXXXX  EQU $8
ooooXXXo  EQU $9
ooo0oooo  EQU $A
ooo1oooo  EQU $B
ooXoXXXX  EQU $C
o1oooooo  EQU $D
```

```
XVXXXXXX  EQU $E
; --
; ----------------------------------------<

; ----------------------------------------> alu_cond_sel vec defines
; --
ZERO      EQU $0
ONE   EQU $1
CCR_C     EQU $2
ASHIFT    EQU $3
; --
; ----------------------------------------<

; ----------------------------------------> usq_cond_sel vec defines
; --
;ZERO     EQU $0
;ONE      EQU $1
;CCR_C    EQU $2
CCR_V     EQU $3
CCR_Z     EQU $4
CCR_N     EQU $5
CCR_BLE       EQU $6
CCR_BLT       EQU $7
CCR_BLS       EQU $8
; --
; ----------------------------------------<

; ----------------------------------------> register_sel vec defines
; --
;ZERO     EQU $0
EA    EQU $1
PC    EQU $2
SP    EQU $3
Y     EQU $4
X     EQU $5
D     EQU $6
B     EQU $7
A     EQU $8
MEM_U16       EQU $9
MEM_U8    EQU $A
MEM_S8    EQU $B
CCR   EQU $C
; --
; ----------------------------------------<

; ----------------------------------------> address_alu_op vec defines
; --
PRE_INC       EQU $0
PRE_INC2  EQU $1
PRE_DEC       EQU $2
PRE_DEC2  EQU $3
POST_INC  EQU $4
POST_INC2 EQU $5
POST_DEC  EQU $6
POST_DEC2 EQU $7
PASS      EQU $8
; --
; ----------------------------------------<

; ----------------------------------------> data_alu_op vec defines
; --
A_PLUS_B  EQU $0
A_PLUS_NOT_B EQU $1
A_AND_B       EQU $2
A_OR_B    EQU $3
A_XOR_B       EQU $4
LSHIFT_A  EQU $5
RSHIFT_A  EQU $6
; --
; ----------------------------------------<

; ----------------------------------------> data_alu_mode vec defines
; --
MODE_8    EQU $0
MODE_16       EQU $1
; --
; ----------------------------------------<

; ----------------------------------------> mem_func_sel vec defines
; --
IDLE      EQU $0
READ_BYTE EQU $1
WRITE_BYTE    EQU $2
READ_WORD EQU $3
WRITE_WORD    EQU $4
READ_OPCODE   EQU $5
; --
; ----------------------------------------<

; --=========================================================================--


; --===========================> High Level Macros <=========================--

; ----------------------------------------> Next Address Marcos
; --
CONTINUE      macro
var_micro_op      set CONTINUE_VEC
          endm

JUMP          macro  \1
var_micro_op      set JUMP_VEC
var_branch_vec        set ((\1)/WORD_WIDTH)&($FF)
          endm

JUMP_MAP0     macro
var_micro_op      set JUMP_MAP0_VEC
          endm

JUMP_MAP1     macro
var_micro_op      set JUMP_MAP1_VEC
          endm

JUMP_MAP2     macro
var_micro_op      set JUMP_MAP2_VEC
          endm

JUMP_MAP3     macro
var_micro_op      set JUMP_MAP3_VEC
          endm

JUMP_MAP4     macro
var_micro_op      set JUMP_MAP4_VEC
          endm

JUMP_MAP5     macro
var_micro_op      set JUMP_MAP5_VEC
          endm
; --
; ----------------------------------------<

; ----------------------------------------> Conditional Macros
; --
IF        macro  \1,\2
var_usq_cond_sel set \1
var_true_false        set \2
          endm
; --
; ----------------------------------------<

; ----------------------------------------> CCR Operation
; --
CCR_OP        macro  \1
var_ccr_op        set \1
```

```
          endm
; --
; ---------------------------------------<

; ---------------------------------------> Address Operations
; --
ADDR_PASS    macro  \1
var_addr_alu_op     set PASS
var_addr_sel    set \1
          endm

ADDR_PRE_INC    macro  \1
var_addr_alu_op     set PRE_INC
var_addr_sel    set \1
          endm

ADDR_PRE_INC2   macro  \1
var_addr_alu_op     set PRE_INC2
var_addr_sel    set \1
          endm

ADDR_PRE_DEC    macro  \1
var_addr_alu_op     set PRE_DEC
var_addr_sel    set \1
          endm

ADDR_PRE_DEC2   macro  \1
var_addr_alu_op     set PRE_DEC2
var_addr_sel    set \1
          endm

ADDR_POST_INC   macro  \1
var_addr_alu_op     set POST_INC
var_addr_sel    set \1
          endm

ADDR_POST_INC2      macro  \1
var_addr_alu_op     set POST_INC2
var_addr_sel    set \1
          endm

ADDR_POST_DEC   macro  \1
var_addr_alu_op     set POST_DEC
var_addr_sel    set \1
          endm

ADDR_POST_DEC2      macro  \1
var_addr_alu_op     set POST_DEC2
var_addr_sel    set \1
          endm
; --
; ---------------------------------------<

; ---------------------------------------> Data Operations
; --
DATA8_PASS      macro  \1
var_data_alu_mode   set MODE_8
var_data_alu_cond_sel  set ZERO
var_data_alu_op     set A_PLUS_B
var_data_a_sel      set \1
var_data_b_sel      set ZERO
          endm

DATA8_ADD    macro  \1,\2
var_data_alu_mode   set MODE_8
var_data_alu_cond_sel  set ZERO
var_data_alu_op     set A_PLUS_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA8_ADDC      macro  \1,\2
```

```
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set CCR_C
var_data_alu_op      set A_PLUS_B
var_data_a_sel       set \1
var_data_b_sel       set \2
          endm

DATA8_SUB    macro  \1,\2
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set ONE
var_data_alu_op      set A_PLUS_NOT_B
var_data_a_sel       set \1
var_data_b_sel       set \2
          endm

DATA8_SUBC      macro  \1,\2
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set CCR_C
var_data_alu_op      set A_PLUS_NOT_B
var_data_a_sel       set \1
var_data_b_sel       set \2
          endm

DATA8_NOT    macro  \1
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set ZERO
var_data_alu_op      set A_PLUS_NOT_B
var_data_a_sel       set ZERO
var_data_b_sel       set \1
          endm

DATA8_AND    macro  \1,\2
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set ZERO
var_data_alu_op      set A_AND_B
var_data_a_sel       set \1
var_data_b_sel       set \2
          endm

DATA8_OR     macro  \1,\2
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set ZERO
var_data_alu_op      set A_OR_B
var_data_a_sel       set \1
var_data_b_sel       set \2
          endm

DATA8_XOR    macro  \1,\2
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set ZERO
var_data_alu_op      set A_XOR_B
var_data_a_sel       set \1
var_data_b_sel       set \2
          endm

DATA8_LSHIFT    macro  \1,\2
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set \2
var_data_alu_op      set LSHIFT_A
var_data_a_sel       set \1
var_data_b_sel       set ZERO
          endm

DATA8_RSHIFT    macro  \1,\2
var_data_alu_mode    set MODE_8
var_data_alu_cond_sel   set \1
var_data_alu_op      set RSHIFT_A
var_data_a_sel       set \2
var_data_b_sel       set ZERO
          endm

DATA8_INC    macro  \1
```

```
var_data_alu_mode   set MODE_8
var_data_alu_cond_sel   set ONE
var_data_alu_op     set A_PLUS_B
var_data_a_sel      set \1
var_data_b_sel      set ZERO
          endm

DATA8_DEC    macro  \1
var_data_alu_mode   set MODE_8
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_PLUS_NOT_B
var_data_a_sel      set \1
var_data_b_sel      set ZERO
          endm

DATA16_PASS      macro  \1
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_PLUS_B
var_data_a_sel      set \1
var_data_b_sel      set ZERO
          endm

DATA16_ADD       macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_PLUS_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA16_ADDC      macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set CCR_C
var_data_alu_op     set A_PLUS_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA16_SUB       macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ONE
var_data_alu_op     set A_PLUS_NOT_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA16_SUBC      macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set CCR_C
var_data_alu_op     set A_PLUS_NOT_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA16_NOT       macro  \1
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_PLUS_NOT_B
var_data_a_sel      set ZERO
var_data_b_sel      set \1
          endm

DATA16_AND       macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_AND_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA16_OR    macro  \1,\2
```

```
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_OR_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA16_XOR       macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_XOR_B
var_data_a_sel      set \1
var_data_b_sel      set \2
          endm

DATA16_LSHIFT    macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set \2
var_data_alu_op     set LSHIFT_A
var_data_a_sel      set \1
var_data_b_sel      set ZERO
          endm

DATA16_RSHIFT    macro  \1,\2
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set \1
var_data_alu_op     set RSHIFT_A
var_data_a_sel      set \2
var_data_b_sel      set ZERO
          endm

DATA16_INC       macro  \1
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ONE
var_data_alu_op     set A_PLUS_B
var_data_a_sel      set \1
var_data_b_sel      set ZERO
          endm

DATA16_DEC       macro  \1
var_data_alu_mode   set MODE_16
var_data_alu_cond_sel   set ZERO
var_data_alu_op     set A_PLUS_NOT_B
var_data_a_sel      set \1
var_data_b_sel      set ZERO
          endm


DATA_WRITE       macro  \1
var_data_wr_sel     set \1
          endm
; --
; ----------------------------------------<

; ----------------------------------------> Memory Controller Function
; --
LOAD_OPCODE      macro
var_mem_func_sel set READ_OPCODE
          endm

LOAD_DATA8       macro
var_mem_func_sel set READ_BYTE
          endm

LOAD_DATA16      macro
var_mem_func_sel set READ_WORD
          endm

STORE_DATA8      macro
var_mem_func_sel set WRITE_BYTE
          endm
```

```
STORE_DATA16     macro
var_mem_func_sel set WRITE_WORD
          endm
; --
; --------------------------------------<


; --============================================================================--


; --=======================> Micro Word Building Macros <====================--

ORGA         macro  \1
          ORG \1*WORD_WIDTH
          endm

default          macro

var_micro_op     set CONTINUE_VEC
var_true_false       set FALSE
var_branch_vec       set $00

var_ccr_op       set oooooooo
var_usq_cond_sel set ZERO
var_data_alu_cond_sel   set ZERO

var_data_a_sel       set ZERO
var_data_b_sel       set ZERO
var_data_wr_sel      set ZERO

var_addr_a_sel       set ZERO
var_addr_b_sel       set ZERO
var_addr_wr_sel      set ZERO

var_addr_alu_op      set PASS

var_data_alu_op      set A_PLUS_B
var_data_alu_mode    set MODE_8

var_mem_func_sel set IDLE

          endm


set_map0  macro
       ; This is processed by map.c
       endm

set_map1  macro
       ; This is processed by map.c
       endm

set_map2  macro
       ; This is processed by map.c
       endm

set_map3  macro
       ; This is processed by map.c
       endm

set_map4  macro
       ; This is processed by map.c
       endm

set_map5  macro
       ; This is processed by map.c
       endm

end_state macro

   dc.b
   ((var_micro_op&$07)<<5)!((var_true_false&$01)<<4)!((var_branch_vec&$F0)>>4)
```

```
    dc.b   ((var_branch_vec&$0F)<<4)!((var_ccr_op&$1E)>>1)
    dc.b
    ((var_ccr_op&$01)<<7)!((var_data_alu_cond_sel&$03)<<5)!((var_usq_cond_sel&$0
F)<<1)!((var_addr_sel&$08)>>3)
    dc.b
    ((var_addr_sel&$07)<<5)!((var_data_a_sel&$0F)<<1)!((var_data_b_sel&$08)>>3)

    dc.b
    ((var_data_b_sel&$07)<<5)!((var_data_wr_sel&$0F)<<1)!((var_addr_alu_op&$08)>
>3)
    dc.b
    ((var_addr_alu_op&$07)<<5)!((var_data_alu_op&$07)<<2)!((var_data_alu_mode&$0
1)<<1)!((var_mem_func_sel&$04)>>2)
    dc.b   ((var_mem_func_sel&$03)<<6)!$00
    dc.b   0

    dc.b   0
    dc.b   0
    dc.b   0
    dc.b   0

    dc.b   0
    dc.b   0
    dc.b   0
    dc.b   0

    nolist
    default
    list

    endm

; --========================================================================--

default
```

```
/*

; --=========================> Gator uProccessor <==========================-
-
; -- File:      MAP.c
; -- Engineer:      Kevin Phillipson
; -- Date:      03.28.07
; -- Revision:      04.01.07
; -- Description:
; --=======================================================================-
-

*/

#include <stdio.h>
#include <string.h>

#define BUF_SIZE 16
#define DEPTH 256

typedef   unsigned char u08;
typedef   signed char  s08;
typedef   unsigned short   u16;
typedef   signed short s16;
typedef   unsigned long u32;
typedef   signed long  s32;

u08 hex2bin(u08 hex_digit);
u08 bin2hex(u08 nibble);
s16 load_buffer(void);
void   crlf(void);
void   quote(void);


u08 buffer[BUF_SIZE];
s16 eof_buf[BUF_SIZE];


main(void)
{

   u08 map_addr;
   s16 idx;
   u08 map_idx;
   s08 flush;
   u08 map_data[6][DEPTH];
   u08 state = 0;

   for (map_idx = 0; map_idx <= 5; map_idx++)
      for (idx = 0; idx < DEPTH; idx++)
         map_data[map_idx][idx] = 0xFF;

   for (idx = 0; idx < BUF_SIZE; idx++)
      buffer[idx] = 0x00;

   for (idx = 0; idx < BUF_SIZE; idx++)
      eof_buf[idx] = 0x00;

   while (load_buffer() != EOF)
   {
      flush = 0;

      switch(state)
      {
         case 0:
            if (buffer[0] == ';')
            {
               flush = 1;
               state = 1;
            }
            if (strncmp(buffer,"END_STATE",9) == 0)
            {
```

```
            flush = 9;
            map_addr++;
         }
         if (strncmp(buffer,"SET_MAP0",8) == 0)
         {
            map_idx = 0;
            flush = 8;
            state = 2;
         }
         if (strncmp(buffer,"SET_MAP1",8) == 0)
         {
            map_idx = 1;
            flush = 8;
            state = 2;
         }
         if (strncmp(buffer,"SET_MAP2",8) == 0)
         {
            map_idx = 2;
            flush = 8;
            state = 2;
         }
         if (strncmp(buffer,"SET_MAP3",8) == 0)
         {
            map_idx = 3;
            flush = 8;
            state = 2;
         }
         if (strncmp(buffer,"SET_MAP4",8) == 0)
         {
            map_idx = 4;
            flush = 8;
            state = 2;
         }
         if (strncmp(buffer,"SET_MAP5",8) == 0)
         {
            map_idx = 5;
            flush = 8;
            state = 2;
         }
         break;

      case 1:
         if ((buffer[0] == 0x0A) || (buffer[0] == 0x0D))
         {
            flush = 1;
            state = 0;
         }
         break;

      case 2:
         if ((buffer[0] == 0x0A) || (buffer[0] == 0x0D))
         {
            flush = 1;
            state = 0;
         }
         if (buffer[0] == ';')
         {
            flush = 1;
            state = 1;
         }
         if ((hex2bin(buffer[0]) != 0xFF) && (hex2bin(buffer[1]) != 0xFF))
         {
            idx = (hex2bin(buffer[0]) << 4) + hex2bin(buffer[1]);

            if (map_data[map_idx][idx] == 0xFF)
            {
               map_data[map_idx][idx] = map_addr;
               flush = 2;
            }
            else
            {
               printf("Error: opcode ");
```

```
                    putchar(bin2hex((idx & 0xF0) >> 4));
                    putchar(bin2hex(idx & 0x0F));
                    printf(" used more than once in map");
                    putchar(bin2hex(map_idx & 0x0F));
                    crlf();
                    //return(0);
                }
            }
            break;

        default:
            state = 0;
            break;
    }

    for (idx = 1; idx < flush; idx++)
        load_buffer();
}


printf("--=========================> Gator uProccessor
<=========================--");crlf();
printf("--");crlf();
printf("-- Engineer:   Kevin Phillipson");crlf();
printf("-- Description:");crlf();
printf("--");crlf();
printf("--
===========================================================================--
");crlf();
printf("");crlf();
printf("library ieee;");crlf();
printf("use ieee.std_logic_1164.all;");crlf();
printf("use ieee.std_logic_unsigned.all;");crlf();
printf("use ieee.std_logic_arith.all;");crlf();
printf("");crlf();
printf("entity mapper is port");crlf();
printf("(");crlf();
printf("  opcode    :  in  std_logic_vector( 7 downto  0);");crlf();
printf("");crlf();
printf("  map0_vector  :  out std_logic_vector( 7 downto  0);");crlf();
printf("  map1_vector  :  out std_logic_vector( 7 downto  0);");crlf();
printf("  map2_vector  :  out std_logic_vector( 7 downto  0);");crlf();
printf("  map3_vector  :  out std_logic_vector( 7 downto  0);");crlf();
printf("  map4_vector  :  out std_logic_vector( 7 downto  0);");crlf();
printf("  map5_vector  :  out std_logic_vector( 7 downto  0)");crlf();
printf("");crlf();
printf(");");crlf();
printf("end mapper;");crlf();
printf("");crlf();
printf("architecture behavior of mapper is");crlf();
printf("");crlf();
printf("begin");
crlf();
crlf();

for (map_idx = 0; map_idx <= 5; map_idx++)
{

    printf("  with opcode select");crlf();
    printf("  map");putchar(bin2hex(map_idx));printf("_vector   <=");crlf();
    for (idx = 0; idx < DEPTH; idx++)
    {

        if((map_data[map_idx][idx] != 0xFF) || (idx == 0))
        {
            printf("     x");quote();

            putchar(bin2hex((map_data[map_idx][idx] >> 4) & 0x0F));
            putchar(bin2hex((map_data[map_idx][idx]) & 0x0F));

            quote();printf(" when   x");quote();
            putchar(bin2hex((idx >> 4) & 0x0F));
            putchar(bin2hex((idx) & 0x0F));

            quote();printf(",");crlf();
         }
      }
      printf("     x");
      quote();
      printf("FF");
      quote();
      printf("  when   others;");
      crlf();
      crlf();
   }

   printf("end behavior;");crlf();

   //return(0);

}

void quote(void)
{
   putchar(0x22);
}

s16 load_buffer(void)
{
   s16 tmp;

   for (tmp = 0; tmp < BUF_SIZE-1; tmp++)
      buffer[tmp] = buffer[tmp+1];

   for (tmp = 0; tmp < BUF_SIZE-1; tmp++)
      eof_buf[tmp] = eof_buf[tmp+1];

   if (eof_buf[BUF_SIZE-1] != EOF)  //
   {
      tmp = getchar();

      //make ucase
      if((tmp >= 0x61) && (tmp <= 0x7A))
         tmp = tmp - 0x20;

   }
   else
   {
      tmp = EOF;
   }

   buffer[BUF_SIZE-1] = tmp;
   eof_buf[BUF_SIZE-1] = tmp;

   return eof_buf[0];
}

void crlf(void)
{
   putchar(0x0D);
   putchar(0x0A);
}

u08 hex2bin(u08 hex_digit)
{
   u08 nibble;

   if ((hex_digit >= 0x30) && (hex_digit <= 0x39))
      nibble = hex_digit - 0x30;
   else
      if ((hex_digit >= 0x41) && (hex_digit <= 0x46))
         nibble = hex_digit - 0x37;
      else
```

```
          nibble = 0xFF;

   return nibble;
}

u08 bin2hex(u08 nibble)
{
   u08 hex_digit;

   if (nibble < 0x0A)
      hex_digit = nibble + 0x30;
   else
      hex_digit = nibble + 0x37;

   return hex_digit;
}
```

```
/*
-----------------------------------------------------------------------------

   Engineer: Kevin Phillipson
   Date:     05/03/2005

-----------------------------------------------------------------------------
*/

#include <stdio.h>

#define DEPTH 256
#define WIDTH 56

typedef   unsigned char u08;
typedef   signed char   s08;
typedef   unsigned short   u16;
typedef   signed short s16;
typedef   unsigned long u32;
typedef   signed long   s32;

u08 hex2bin(u08 hex_digit);
u08 bin2hex(u08 nibble);
void crlf(void);

main(void)
{

   u16    mif_addr;
   s16    tmp;
   u08    data[DEPTH][(WIDTH/4)];
   u08    check_sum;
   u08    byte;

   for (mif_addr = 0; mif_addr < DEPTH; mif_addr++)
      for (tmp = 0; tmp < (WIDTH/4); tmp++)
         data[mif_addr][tmp] = '0';

   while ((tmp = getchar()) != EOF)
   {
      byte = tmp;
      if (byte == 'S')
         if (getchar() == '2')
         {
            getchar();
            getchar();
            getchar();

            mif_addr = 0;
            for (tmp = 3; tmp >= 0; tmp--)
               mif_addr += hex2bin(getchar()) << (4 * tmp);

            getchar();

            for (tmp = 0; tmp < (WIDTH/4); tmp++)
               data[mif_addr][tmp] = getchar();
         }
   }

   printf("DEPTH         = %d;", DEPTH); crlf();
   printf("WIDTH         = %d;", WIDTH); crlf();
   printf("ADDRESS_RADIX = HEX;"); crlf();
   printf("DATA_RADIX    = HEX;"); crlf(); crlf();

   printf("CONTENT"); crlf();
   printf("BEGIN"); crlf(); crlf();

   for (mif_addr = 0; mif_addr < DEPTH; mif_addr++)
   {
      check_sum = 0;
```

```
        for (tmp = 3; tmp >= 0; tmp--)
        {
           byte = (mif_addr >> (4 * tmp)) & 0x0F;

           if (tmp & 0x01)
              check_sum += byte << 4;
           else
              check_sum += byte;

           putchar(bin2hex(byte));
        }

        printf(" : ");

        for (tmp = 0; tmp < (WIDTH/4); tmp++)
        {
           byte = data[mif_addr][tmp];

           if (tmp & 0x01)
              check_sum += hex2bin(byte);
           else
              check_sum += hex2bin(byte) << 4;

           putchar(byte);
        }
        printf(";");crlf();
     }

     crlf();
     printf("END;"); crlf();

  }

  void crlf(void)
  {
     putchar(0x0D);
     putchar(0x0A);
  }

  u08 hex2bin(u08 hex_digit)
  {
     u08 nibble;

     if ((hex_digit >= 0x30) && (hex_digit <= 0x39))
        nibble = hex_digit - 0x30;
     else
        if ((hex_digit >= 0x41) && (hex_digit <= 0x46))
           nibble = hex_digit - 0x37;
        else
           nibble = 0xFF;

     return nibble;
  }

  u08 bin2hex(u08 nibble)
  {
     u08 hex_digit;

     if (nibble < 0x0A)
        hex_digit = nibble + 0x30;
     else
        hex_digit = nibble + 0x37;

     return hex_digit;
  }
```

```
/*

; --=========================> Gator uProccessor <==========================-
-
; -- File:      s192mif8.c
; -- Engineer:      Kevin Phillipson
; -- Date:      03.28.07
; -- Revision:      04.01.07
; -- Description:
; --=========================================================================-
-

*/

#include <stdio.h>
#include <string.h>

#define WIDTH 8

#define BUF_SIZE 16

typedef   unsigned char u08;
typedef   signed char   s08;
typedef   unsigned short   u16;
typedef   signed short  s16;
typedef   unsigned long u32;
typedef   signed long   s32;

u08 hex2bin(u08 hex_digit);
u08 bin2hex(u08 nibble);
s16 load_buffer(void);
void   crlf(void);


u08 buffer[BUF_SIZE];
s16 eof_buf[BUF_SIZE];


main(void)
{


   u32 idx = 0;

   u08 num_bytes;

   u16 mif_addr;
   u32 mif_depth = 0;

   u32 raw_idx = 0;
   u16 raw_addr[0x10000];
   u08 raw_data[0x10000];

   s08 flush;
   u08 state = 0;

   for (idx = 0; idx < BUF_SIZE; idx++)
      buffer[idx] = 0x00;

   for (idx = 0; idx < BUF_SIZE; idx++)
      eof_buf[idx] = 0x00;

   while (load_buffer() != EOF)
   {
      flush = 0;

      switch(state)
      {
         case 0:
            if (strncmp(buffer,"S1",2) == 0)
            {
               num_bytes =  (hex2bin(buffer[2]) << 4) +
                    (hex2bin(buffer[3])) - 3;

                mif_addr =   (hex2bin(buffer[4]) << 12) +
                    (hex2bin(buffer[5]) << 8) +
                    (hex2bin(buffer[6]) << 4) +
                    (hex2bin(buffer[7]));

                flush = 8;
                state = 1;
            }
            break;

        case 1:
            if (num_bytes != 0)
            {
                if (mif_addr > mif_depth)
                {
                    mif_depth = mif_addr;
                }

                raw_addr[raw_idx] = mif_addr;
                raw_data[raw_idx] =    (hex2bin(buffer[0]) << 4) +
                          (hex2bin(buffer[1]));

                raw_idx++;
                mif_addr++;
                num_bytes--;
                flush = 2;

                if (num_bytes == 0)
                {
                    state = 0;
                }
            }
            else
            {
                state = 0;
            }

            break;

        default:
            state = 0;
            break;
    }

    for (idx = 1; idx < flush; idx++)
        load_buffer();
}

printf("DEPTH = %d;", (mif_depth+1));crlf();
printf("WIDTH = %d;", WIDTH);crlf();
printf("ADDRESS_RADIX = HEX;");crlf();
printf("DATA_RADIX = HEX;");crlf();
printf("");crlf();
printf("CONTENT");crlf();
printf("BEGIN");crlf();
printf("");crlf();

for (idx = 0; idx < raw_idx; idx++)
{

    putchar(bin2hex((raw_addr[idx] >> 12) & 0x0F));
    putchar(bin2hex((raw_addr[idx] >> 8) & 0x0F));
    putchar(bin2hex((raw_addr[idx] >> 4) & 0x0F));
    putchar(bin2hex((raw_addr[idx] >> 0) & 0x0F));

    printf(" : ");

    putchar(bin2hex((raw_data[idx] >> 4) & 0x0F));
    putchar(bin2hex((raw_data[idx] >> 0) & 0x0F));
```

```
      printf(";");crlf();

   }

   printf("");crlf();
   printf("END;");crlf();

   //return(0);

}


s16 load_buffer(void)
{
   s16 tmp;

   for (tmp = 0; tmp < BUF_SIZE-1; tmp++)
      buffer[tmp] = buffer[tmp+1];

   for (tmp = 0; tmp < BUF_SIZE-1; tmp++)
      eof_buf[tmp] = eof_buf[tmp+1];

   if (eof_buf[BUF_SIZE-1] != EOF)  //
   {
      tmp = getchar();

      //make ucase
      if((tmp >= 0x61) && (tmp <= 0x7A))
         tmp = tmp - 0x20;

   }
   else
   {
      tmp = EOF;
   }

   buffer[BUF_SIZE-1] = tmp;
   eof_buf[BUF_SIZE-1] = tmp;

   return eof_buf[0];
}

void crlf(void)
{
   putchar(0x0D);
   putchar(0x0A);
}

u08 hex2bin(u08 hex_digit)
{
   u08 nibble;

   if ((hex_digit >= 0x30) && (hex_digit <= 0x39))
      nibble = hex_digit - 0x30;
   else
      if ((hex_digit >= 0x41) && (hex_digit <= 0x46))
         nibble = hex_digit - 0x37;
      else
         nibble = 0xFF;

   return nibble;
}

u08 bin2hex(u08 nibble)
{
   u08 hex_digit;

   if (nibble < 0x0A)
      hex_digit = nibble + 0x30;
   else
      hex_digit = nibble + 0x37;
```

```
    return hex_digit;
}
```

```
*
* Gator uProcessor S19 Bootloader
*
* Kevin Phillipson
*
    .nolist
    .list

UART_BUFFER   EQU $8000
UART_STATUS   EQU $8001
LEDS      EQU $FFFF
PROG_START    EQU $0100

    ORG $0000

    LDS #$0FFF

    LDX #PROMPT
    JSR UART_PRINTX
    CLR TOTAL_BYTES
    CLR TOTAL_BYTES_LO

MAIN_LOOP

    JSR UART_GETA

    CMPA   #'S'
    BNE MAIN_LOOP

    JSR UART_GETA

    CMPA   #'1'
    BEQ S1_RECORD

    CMPA   #'9'
    BEQ S9_RECORD

    JMP MAIN_LOOP

S9_RECORD
    LDX #LOAD_STATUS
    JSR UART_PRINTX
    LDAA   TOTAL_BYTES
    JSR PUTHEX_BYTE
    LDAA   TOTAL_BYTES_LO
    JSR PUTHEX_BYTE
    JMP PROG_START

S1_RECORD

    JSR GETHEX_BYTE
    SUBA   #3
    STAA   BYTE_CNT
    TAB
    LDX TOTAL_BYTES
    ABX
    STX TOTAL_BYTES
    JSR GETHEX_BYTE
    STAA   WRITE_ADDR
    JSR GETHEX_BYTE
    STAA   WRITE_ADDR_LO


S1R_LOOP
    LDAA   BYTE_CNT
    BEQ S1R_END
    DECA
    STAA   BYTE_CNT
    JSR GETHEX_BYTE
    LDX WRITE_ADDR
    STAA   0,X
    INX
    STX WRITE_ADDR
    JMP S1R_LOOP

S1R_END
    JMP MAIN_LOOP

*------------------------------>
PUTHEX_BYTE
    PSHA

    LSRA
    LSRA
    LSRA
    LSRA
    JSR PUTHEX

    PULA
    PSHA
    ANDA   #$0F
    JSR PUTHEX

    PULA
    RTS
*------------------------------<

*------------------------------>
PUTHEX
    PSHA
    CMPA   #$0A
    BLO PUTHEX1
    ADDA   #$37
    JMP PUTHEX2
PUTHEX1
    ADDA   #$30
PUTHEX2
    JSR UART_PUTA
    PULA
    RTS
*------------------------------<

*------------------------------> Get 2 hex characters and convert to binary
byte. Store in A
GETHEX_BYTE
    PSHB
    JSR GETHEX
    LSLA
    LSLA
    LSLA
    LSLA
    TAB
    JSR GETHEX
    ABA
    PULB
    RTS
*------------------------------<

*------------------------------> Get hex character and convert to binary nibble.
Store in A
GETHEX
    JSR UART_GETA
    CMPA   #$41
    BHS GETHEX1
    SUBA   #$30
    JMP GETHEX2
GETHEX1
    SUBA   #$37
GETHEX2
    RTS
*------------------------------<

*------------------------------> Recieve byte and place in A
UART_GETA
```

```
    LDAA   UART_STATUS
    ANDA   #$01
    BEQ UART_GETA
    LDAA   UART_BUFFER
*   JSR UART_PUTA
    RTS
*-----------------------------<

*-----------------------------> Send byte contained in A
UART_PUTA
    PSHA
UPA_WAIT
    LDAA   UART_STATUS
    ANDA   #$02
    BNE UPA_WAIT
    PULA
    STAA   UART_BUFFER
    RTS
*-----------------------------<

*-----------------------------> Print null terminated string pointed to by X
UART_PRINTX
    PSHA
UPX_LOOP
    LDAA   0,X
    CMPA   #$00
    BEQ UPX_END
    JSR UART_PUTA
    INX
    JMP UPX_LOOP
UPX_END
    PULA
    RTS
*-----------------------------<

TOTAL_BYTES   DC.B   $00
TOTAL_BYTES_LO   DC.B   $00
BYTE_CNT  DC.B   $00
WRITE_ADDR   DC.B   $00
WRITE_ADDR_LO DC.B   $00

PROMPT
    DC.B   $0D
    DC.B   $0A
    DC.B   $0D
    DC.B   $0A
    DC.B   "Gator uProcessor S19 Bootloader..."
    DC.B   $00

LOAD_STATUS
    DC.B   $0D
    DC.B   $0A
    DC.B   "Bytes loaded: $"
    DC.B   $00
```