2004

The Effects of Instruction Set Variation on a Reconfigurable Processor

Ira Harmon
University of North Florida

The Effects of Instruction Set Variation on a Reconfigurable Processor

Ira Harmon

Faculty Sponsor: Dr. Chiu Choi, Associate Professor of Electrical Engineering

Abstract

The emerging field of reconfigurable computing promises increased processing power in terms of speed and function with only a modest investment in hardware. The goal of this experiment was to test this premise. Using a FPGA, a CPU was implemented in Verilog HDL. The CPU was then tested to measure the improvement of program execution time through the use of hardwired instructions in a reconfigurable system. Secondly, the predicted gain in reconfigurable resources as function of the number of instructions was tested in a reconfigurable environment. The results of the experiment suggest reconfigurable computing can provide marginal changes in resources allocated and significant improvements in program execution time.

Introduction

Though it is thought of as one of the benchmarks of modern technology, the concept of the computer can be traced back to the 1600s. But since its conception, every incarnation of the man-made counting machine has been stricken by the technological limitations of its age. Machines based on gears, relays, and vacuum tubes have all been imperfect embodiments of the computer, unable to capture the true spirit of Pascal's idea. Today that spirit has been reincarnated in a silicon shell, whose progeny sits atop desks in one out of every two American homes. And though it may seem that the brainchild that took shape in the fertile womb of mathematician's mind has finally come of age, the brilliant glow of Pascal's idea is still lost in the opaque shell of silicon known as the integrated circuit.

For all their complexity, modern CPUs are primarily composed of a single electronic device, the transistor. While the invention of the integrated circuit has allowed millions of transistors to be crammed into a square centimeter of silicon, there is still a limit to the number of these devices that can fit on a chip. Why is this of any importance? Since the transistor is the basis of computer functionality, a limit in the number of transistors limits the functions the chip can implement. Though programming techniques can stretch these limitations, by leaving the rigid mold of modern computer architecture the limits can be broken.

For example, imagine that a CPU with 10 instructions has been sold to a programmer. But the programmer is writing a program that needs an instruction not supported by the CPU. He could write a subroutine, but for complex instructions, such as multiplication, this could cause a large increase in program execution time. However, suppose there are three instructions implemented in the CPU that the programmer will not use. What if he could take the circuitry for the three unused instructions and use it to implement the instruction he needs.

This is the idea behind reconfigurable computing.

Theory

For hardware to be reconfigurable the standard integrated circuit is of little use. Fortunately, the field programmable gate array is capable of morphing itself into various hardware by routing smaller logic blocks. Figure 3.1 is a block diagram of a standard FPGA, Xilinx's XCV100. Only the workings of the inner components as they relate to the parameters of the experiment will be covered here. The majority of the circuitry on the chip is devoted to routing and CLB's. CLBs are configurable logic blocks. Two CLBs form a slice. Inside each CLB is a 4-input function generator that's capable of producing logic for any 4-bit Boolean function. The function generator is composed of a smaller building block, the 4-input look-up table or 4-input LUT. CLBs are connected through the general routing matrix or GRM. The GRM is an array of routing switches. The blocks labeled DLL and VersaRing are both parts of the GRM. The IOBs or input-output blocks are responsible for implementing the input and output functions of the FPGA.

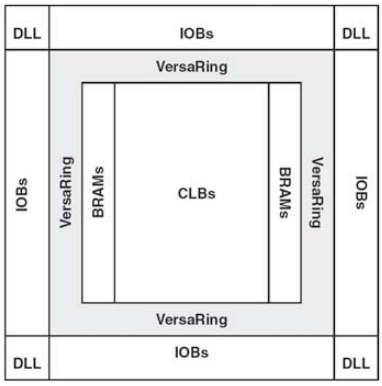

Figure 3.1

The XCV100 has 1200 slices, 2400 4-input LUTs, 170 IOBs, and 4 clocks. Each slice contains two flip-flops for a total of 2400 slice flip-flops. These are the primary resources of the FPGA. The percentage of resources used in creating a design is calculated by Xilinx software.

In this experiment, the XCV100 FPGA is used to implement a single instruction stream single data stream 16-bit RISC processor with 29 instructions, the y42. (A summary of the instruction set is listed in the appendix.) The programmer's model of y42 is shown below in figure 3.2. The flow of data

through the registers and ALU is further expanded in figure 3.3

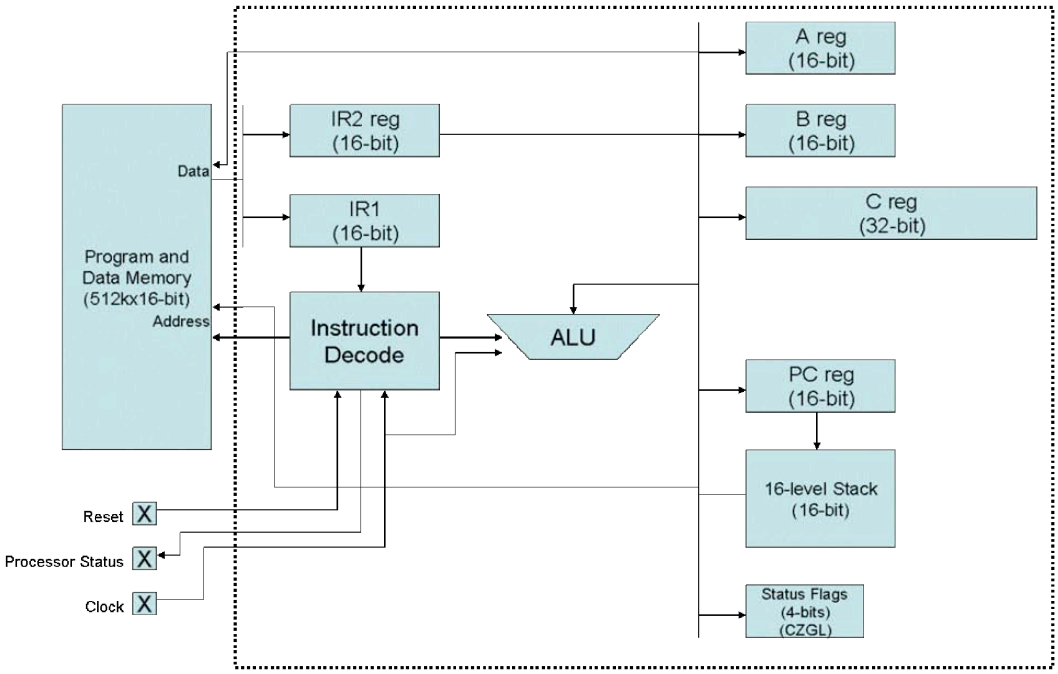

Figure 3.2

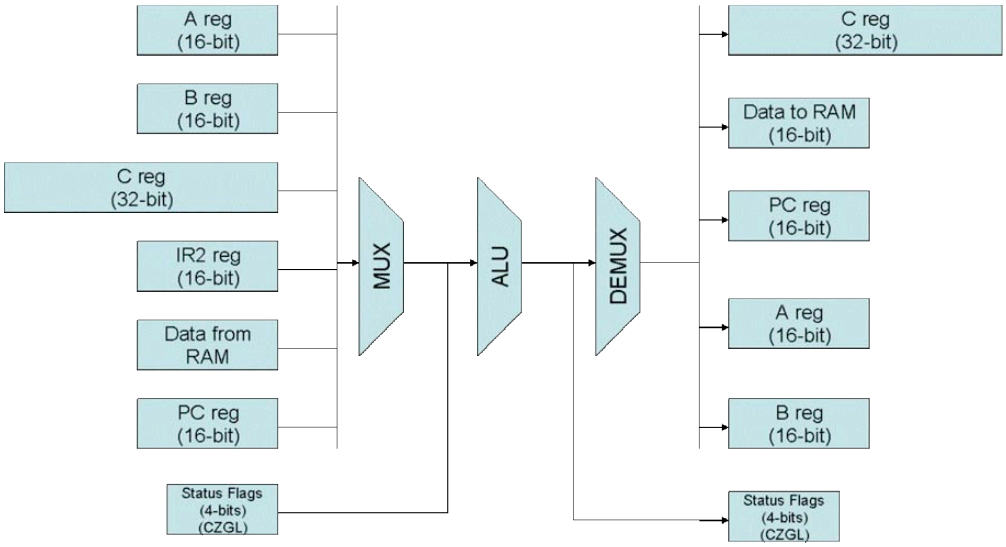

Figure 3.3

Each instruction executes in nine cycles of the clock. During an instruction cycle, two bytes of memory are fetched from external RAM. The two memory fetches in addition to extra cycles needed for the clocked ALU and data buses to load the current data contribute to the unusually large number of clock pulses per executed instruction. Figure 3.4 summarizes the operations that occur during each pulse of an instruction cycle.

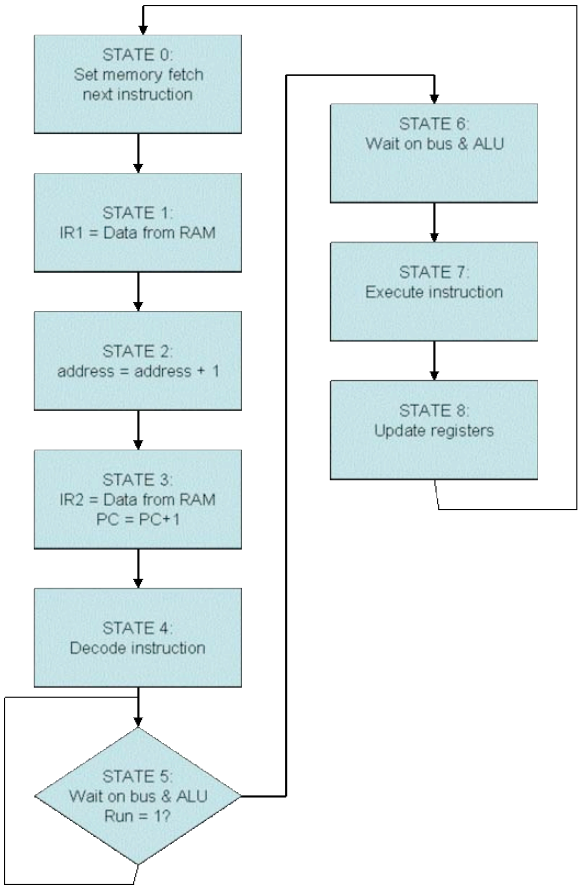

Figure 3.4

Because a hardwired instruction executes in 9 clock cycles, a program containing N-instructions should execute in 9N clock cycles. In short, this is the reason a hardwired instruction out performs software emulated instructions.

Equipment

XSV Board v1.01
Xstools v4.0.2 software suite
Xilinx XST v5.21
ModelSim XE 2 5.6e
Oscilloscope
9VDC power supply
DB-25 printer cable
PC with Window's XP

Procedure

Having started Xilinx Project Navigator a new project was created. The module i/o-ports for the y42 processor included a 19-bit address bus, a 16-bit bidirectional data bus, an output enable bit, a write enable bit, a chip select bit, a system clock bit, a reset bit, and a processor status bit.

The Verilog code describing the operation of the CPU was entered. (The code can be found in the appendix.) Next, an arithmetic logic unit module was added to the y42 project. The ALU operations included were 16-bit addition and subtraction, 8-bit multiplication, 16-

bit logic “and”, logic “or”, logic “not”, and “exclusive or”, 16-bit increment and decrement, shifts right and left, and an instruction to pass the input through the ALU unchanged. In describing the hardware for multiplication and “exclusive or” care was taken to ensure that the hardware produced would operate similarly to its software counterpart.

The data bus was constructed using two modules, a multiplexer that routes registers A, B, C, PC, IR2, and data from the external RAM into the ALU and a demultiplexer that routes the output from the ALU into registers A, B, C, PC, or the data going into the external RAM. Before the project could be compiled a user constraint file or UCF was added to the project. The pins assigned to each port of the processor were chosen appropriately. (See the appendix for the listing of the UCF file.) Finally, after many cycles of simulation and debugging, the y42 project and its modules were compiled and a “bit” file was generated.

Then eight programs were written, “hardware_mult”, “hardware_multT”, “hardware_xor”, “hardware_xorT”, “software_mult”, “software_multT”, “software_xor”, and “software_xorT”. (See the appendix for the assembly and binary code.) The programs were stored as text files. Each assembly language program was hand assembled into binary code. The four programs with the T suffix included instructions to toggle the processor status bit at the beginning and end of the program as well as code that caused the program to run in an infinite loop. The other four programs were used to check for correct program and processor operation.

Next power was applied to the XSV board and the DB-25 cable was connected to the PC printer port.

Before testing began, the clock on the XSV board was set at 5MHz. After the board was prepared the binary file “hardware_mult.XES” was loaded into RAM on the XSV board using Xstool’s GXSload. Also using GXSload, the file “y42processor.bit” was loaded onto the XCV100 FPGA.

The reset switch was lowered then raised and the contents of the XCV board’s.

RAM was downloaded to the PC. The results were checked for accuracy and the process was repeated for each program.

Next channel one of an oscilloscope was connected between the processor status pin and ground. One of the timing programs was loaded into memory and executed. By means of oscilloscope the execution time of the program was measured and recorded. This process was repeated for each program.

To measure the change in FPGA resources used as a function of hardwired instructions, Xilinx Project Navigator was opened. The original CPU was compiled and the percentage of FPGA resources used and the maximum clock speed were noted. Then, the instruction for multiplication was removed and the program was recompiled. (Removal of an instruction consisted of its removal from y42_alu.v and from the decoding portion of y42processoer.v.) After recording the resources and maximum clock speed, the CPU was compiled again, but with the multiplication instruction in place and the “exclusive or” instruction removed. Once again the results of the compilation were observed and recorded. The

program was compiled two more times with the remaining permutations of the removal of either instruction. The results of each compilation were recorded.

Results and Discussion

Figure 6.1 shows the results of the timing data collected for the two programs. As expected the inclusion of hardware encoded instructions increased execution time by more than 100% in both cases.

Program	**Execution Time (uS)**	**Improvement**
software_multT	30	
Hardware_multT	10	3X
software_xorT	104	
Hardware_xorT	9	11.55X

Figure 6.1

However, the removal of hardware encoded instructions had surprisingly little effect on the percentage of FPGA resources used. The inclusion of the multiplier had the greatest effect here, causing a 6% increase in the number of LUTs required. Figure 6.2 summarizes the results.

	XOR	**!XOR**	**DIFFERENCE**
MUL	53% slices 25% slice FFs 47% 4-input LUTs 24% bonded IOBs 25% clocks max. freq. 22.760MHz	52% slices 25% slice FFs 46% 4-input LUTs 24% bonded IOBs 25% clocks max. freq. 21.622MHz	1% slices 0% slice FFs 1% 4-input LUTs 0% bonded IOBs 0% clocks max. freq. 1.138MHz
!MUL	48% slices 25% slice FFs 41% 4-input LUTs 24% bonded IOBs 25% clocks max. freq. 23.173MHz	49% slices 25% slice FFs 41% 4-input LUTs 24% bonded IOBs 25% clocks max. freq. 22.717MHz	-1% slices 0% slice FFs 0% 4-input LUTs 0% bonded IOBs 0% clocks max. freq 0.456MHz
DIFFERENCE	5% slices 0% slice FFs	3% slices 0% slice FFs	

	6% 4-input LUTs 0% bonded IOBs 0% clocks max. freq. -0.413MHz	5% 4-input LUTs 0% bonded IOBs 0% clocks max. freq. -1.095MHz

Figure 6.2

Although the improvement in program execution speed with the inclusion of the appropriate hardwired instruction was expected, the small difference in the FPGA resources consumed, as a function of the instruction set is somewhat more difficult to explain. Because the routine for multiplication was the larger of the two, its absence made a larger difference, but the change in the maximum frequency of the processor was similar for both instructions, suggesting that the speed may be independent of both instructions.

Overall, the marginal change in resources used may be caused by the inclusion of a large number of other instructions. Buried in the large amount of logic required for the entire instruction set, the absence or inclusion of a single instruction probably had little effect. However, the indication of any change was promising because as many instructions as necessary could be removed to to create room for one or more required instructions.

Conclusion

In spite of the marginal gains in allocated FPGA resources as a function of the instruction set, the results were promising nonetheless. The improvement in execution time alone may make the use of reconfigurable computing worthwhile in applications calling for uncommon instructions that are rarely implemented on non-reconfigurable processors.

In future studies it would be interesting to compare reconfigurable computing to emulated processors and virtual machine languages such as Java and explore the possibility of reconfigurable processors as a replacement for either of the two.

However, for reconfigurable computing to become a main stay of the computer industry improved platforms of implementation must be developed. While FPGAs are reasonable platforms for reconfigurable processor development, they are currently cost prohibitive and inefficient in terms of hardware versus functionality. Only with the introduction of more cost effective transistor efficient configurable hardware can the spirit of reconfigurable computer become a reality.

References

Bhasker, J. A Verilog HDL Primer. Allentown: Star Galaxy Press, 1997.

Gordon, Mark A. Verilog Digital Computer Design: Algorithms into Hardware. New Jersey: Prentice Hall PTR, 1999.

Hill, Fredrick J. and Gerald Peterson. Digital Systems: Hardware Organization and Design. 3rd ed. New York: John Wiley and Sons, 1987.

Shiva, Sajjan G. Computer Design and Architecture. 3rd ed. New York: Marcel Dekker, Inc., 2000.

Truss, John. Discrete Mathematics for Computer Scientist. Harlow: Addison Wesley, 1999.

Van den Bout, Dave. The Practical Xilinx Designer Lab Book. New Jersey: Prentice Hall, 1998.

Xess. XSV Board Version 1.0. North Carolina: Xess, 2000.

Xilinx. Virtex 2.5V Field Programmable Gate Arrays. San Jose: Xilinx, 2001.

Xilinx. Xst User Guide. San Jose: Xilinx, 2003.

Appendix

y42processor.v

```
module y42processor (ram_data, address, clk, rst, write_en,
out_en, chip_sel, start, finish);

//start = processor status bit
//XSV100 board clock = 5MHz
   input          clk, rst;
   inout [15:0]  ram_data;

   output [18:0]   address;
   output          write_en, out_en, chip_sel, start, finish;

   reg [3:0]      flags;            //Carry, Zero, Greater than,
Less than
   reg [15:0]     A, B, PC;
   reg [31:0]     C;
   reg [15:0]     IR1, IR2;
   reg [18:0]     address;
   reg            write_en, out_en, chip_sel, PC_alt_cmnd_ex,
flags_alt_cmd_ex, start, finish, run;
   reg [3:0]      oper_decode;
   reg [3:0]      input_sel;
   reg [2:0]      dest_sel;
   reg [3:0]      state;

   reg [15:0]     ram_data_storage;

   reg [3:0]      SP;
   reg [15:0]     stack[15:0];

   wire [3:0]      temp_flags;
   wire [31:0]     mux1_out;
   wire [31:0]     alu_out, temp_C;
   wire [15:0]     temp_PC, temp_A, temp_B, ram_data,
temp_ram_data_storage;

   reg             rw;

   into_alu_mux  mux1(A, B, C, IR2, ram_data, PC,
input_sel, mux1_out, clk);
      //into_alu_mux (in1, in2, in3, in4, in5, in6, ch_sel, out1)

   y42_alu    alu1(mux1_out[31:16], mux1_out[15:0],
flags[3], oper_decode,
                   alu_out, temp_flags[3], temp_flags[2],
temp_flags[1], temp_flags[0], clk);
      //(input1, input2, carry, oper_sel, output1, Cflag, Zflag,
Gflag, Lflag)

   out_alu_demux   demux1(alu_out, dest_sel, temp_C,
temp_ram_data_storage, temp_PC, temp_A, temp_B, clk);
   //out_alu_demux(in1, ch_sel, out1, out2, out3, out5, out6)


   assign ram_data = rw? ram_data_storage : {16{1'bz}};

   always @(posedge clk or posedge rst)
   begin
     if (rst == 1'B1)
       begin
         PC             = 16'B0;
         SP             =  4'B0;
         ram_data_storage  = 16'B0;
         write_en       = 1;
         out_en         = 1;
         chip_sel       = 1;
         state          = 4'B0000;
         rw             = 0;          //0 - input, 1 - output//
         run            = 1;
         start          = 0;
         finish         = 0;
       end
     else
//FETCH
      case (state)
        4'B0000:  begin
                    chip_sel       = 0;
                    PC_alt_cmnd_ex  = 0;
                    flags_alt_cmd_ex = 0;
                    address      = {3'B0,2*PC};
                    out_en         = 0;
                    state          = 4'B0001;
                    rw             = 0;
                  end

        4'B0001: begin
                    IR1        = ram_data;
                    state          = 4'B0010;
                  end
         4'B0010: begin
                    address        = {3'B0, (address + 19'B1)};
                    state          = 4'B0011;
                  end
         4'B0011: begin
                    IR2            = ram_data;
                    PC             = PC + 16'B1;
                    state          = 4'B0100;
                  end
//DECODE
          4'B0100:  begin
                      out_en         = 1;
                      case (IR1[15:11])
```

```
                5'B00001: begin
        //ADD A+B=C
                            input_sel   = 4'B0000;
                            oper_decode = 4'B0000;
                            dest_sel    = 3'B000;
                          end
                5'B00010: begin
        //SUB A-B=C
                            input_sel   = 4'B0000;
                            oper_decode = 4'B0001;
                            dest_sel    = 3'B000;
                          end
                5'B00011: begin
        //MUL A*B=C
                            input_sel   = 4'B0000;
                            oper_decode = 4'B0010;
                            dest_sel    = 3'B000;
                          end
                5'B00100: begin
        //RTS return from sub-routine
                            SP = SP - 1;
                            PC = stack[SP];
                            dest_sel    = 3'B111;
                            PC_alt_cmnd_ex = 0;
                          end

                5'B00101: begin
        //AND A&B=C
                            input_sel   = 4'B0000;
                            oper_decode = 4'B0100;
                            dest_sel    = 3'B000;
                          end
                5'B00110: begin
        //OR A|B=C
                            input_sel   = 4'B0000;
                            oper_decode = 4'B0101;
                            dest_sel    = 3'B000;
                          end
                5'B00111: begin
        //NOT !A=C, !B=C
                            if (IR1[10:9] == 2'B00)
                              input_sel = 4'B0110;
                            else
                              input_sel = 4'B0111;
                            oper_decode  = 4'B0110;
                            dest_sel     = 3'B000;
                          end
                5'B01000: begin
        //EXOR C = A ^ B
                            input_sel   = 4'B0000;
                            oper_decode = 4'B0111;
                            dest_sel    = 3'B000;
                          end
                5'B01001: begin
        //CMP A-B, A-k, B-k
                            if (IR1[10:9] == 2'B00)
                              input_sel   = 4'B0000;
                            else
                              if (IR1[10:9] == 2'B01)
                                input_sel = 4'B0001;
                              else
                                if (IR1[10:9] ==
2'B10)
                                  input_sel =
4'B0010;
                            oper_decode = 4'B0001;
                            dest_sel    = 3'B111;
                          end
                5'B01011: begin
        //LDA  A <= (M), A <= k
                            if (IR1[10:0] == 11'B0)
                              begin
                                address     = {3'B0,
IR2};
                                input_sel   =
4'B0101;
                                rw          = 0;
                                out_en      = 0;
                              end
                            else
                              if (IR1[10:0] ==
11'B01000000000)
                                input_sel   =
3'B0011;
                            oper_decode = 4'B1011;
                            dest_sel  = 3'B100;
                          end
                5'B01100: begin
        //LDB B <= (M), B <=k
                            if (IR1[10:0] == 11'B0)
                              begin
                                address     = {3'B0,
IR2};
                                input_sel   =
4'B0101;
                                rw          = 0;
                                out_en      = 0;

                              end
                            else
                              if (IR1[10:0] ==
11'B01000000000)
                                input_sel   =
3'B0011;
                            oper_decode = 4'B1011;
                            dest_sel  = 3'B101;
                          end
                5'B01101: begin
        //STR (M)<= A, (M)<= B, (M) <= LoC
                            if (IR1[10:0] == 11'B0)
                              input_sel = 4'B0110;
                            else
                              if (IR1[10:0] ==
11'B01000000000)
                                input_sel =
4'B0111;
                              else
                                input_sel = 4'B1001;
                            oper_decode = 4'B1011;
                            address  = {3'B00, IR2};
                            dest_sel = 3'B001;
                            write_en = 0;
                            rw       = 1;
                          end
                5'B01110: begin
        //MOVC   A <= (LoC), B <= (LoC), A:B <= C
                            input_sel   = 4'B1001;
                            oper_decode = 4'B1011;
                            if (IR1[10:9] == 2'B00)
                              dest_sel = 3'B100;
                            else
                              if (IR1[10:9] == 2'B01)
                                dest_sel = 3'B101;
                              else
                                if (IR1[10:9] ==
2'B10)
                                  dest_sel = 3'B110;
                          end
                5'B01111: begin
        //JMP k
```

```
                                    input_sel      = 4'B0011;
                                    oper_decode    =
4'B1011;
                                    dest_sel       = 3'B010;
                                    PC_alt_cmnd_ex = 1;
                                  end
                        5'B10000: begin
  //BEQ k
                                    if (flags[2])
                                      begin
                                        input_sel      =
4'B0011;
                                        oper_decode    =
4'B1011;
                                        dest_sel       =
3'B010;
                                        PC_alt_cmnd_ex =
1;
                                      end
                                    else
                                      oper_decode      =
4'B1111;
                                  end
                        5'B10001: begin
  //BNE k
                                    if (!flags[2])
                                      begin
                                        input_sel      =
4'B0011;
                                        oper_decode    =
4'B1011;
                                        dest_sel       =
3'B010;
                                        PC_alt_cmnd_ex =
1;
                                      end
                                    else
                                      oper_decode      =
4'B1111;
                                  end
                        5'B10010: begin
  //BGT k
                                    if (flags[1])
                                      begin
                                        input_sel      =
4'B0011;
                                        oper_decode    =
4'B1011;
                                        dest_sel       =
3'B010;
                                        PC_alt_cmnd_ex =
1;
                                      end
                                    else
                                      oper_decode      =
4'B1111;
                                   end
                        5'B10011: begin
  //BLT k
                                    if (flags[0])
                                      begin
                                        input_sel      =
4'B0011;   //place here so that flags are changed after jmp
cmds
                                        oper_decode    =
4'B1011;
                                        dest_sel       =
3'B010;
                                        PC_alt_cmnd_ex =
1;
                                      end
                                    else
                                      oper_decode      =
4'B1111;
                                  end
                        5'B10100: begin
  //BSET A,{15..0}; BSET B{15..0}
                                    if (IR1[10:9] == 2'B00)
                                      input_sel = 4'B0001;
                                    else
                                      input_sel = 4'B0010;
                                    oper_decode  = 4'B0101;
                                    if (IR1[10:9] == 2'B00)
                                      dest_sel = 3'B100;
                                    else
                                      dest_sel = 3'B101;
                                  end
                        5'B10101: begin                 //BCLR
A, {15..0}; BCLR B, {15..0}
                                    if (IR1[10:0] == 11'B00)
                                      input_sel = 4'B0001;
                                    else
                                      input_sel = 4'B0010;
                                    oper_decode  = 4'B1100;
                                    if (IR1[10:9] == 2'B00)
                                      dest_sel = 3'B100;
                                    else
                                      dest_sel = 3'B101;

                                  end
                        5'B10110: begin                 //CLC
Carry <= 0
                                    flags[3]         = 0;
                                    flags_alt_cmd_ex  = 1;
                                    dest_sel         = 3'B111;
                                  end
                        5'B10111: begin                 //SEC
Carry <= 1
                                    flags[3]        = 1;
                                    flags_alt_cmd_ex = 1;
                                    dest_sel         = 3'B111;
                                  end
                        5'B11000: begin                 //SHL
C <= A << 1, C <= B << 1
                                    if (IR1[10:9] == 2'B00)
                                      input_sel = 4'B0110;
                                    else
                                      input_sel = 4'B0111;
                                    oper_decode  = 4'B1110;
                                    dest_sel = 3'B000;
                                  end
                        5'B11001: begin                 //SHR  C
<= A >> 1, C <= B >> 1
                                    if (IR1[10:9] == 2'B00)
                                      input_sel = 4'B0110;
                                    else
                                      input_sel = 4'B0111;
                                    oper_decode  = 4'B1101;
                                    dest_sel   = 3'B000;
                                  end
                        5'B11010: begin                 // JSR  k
jump sub-routine
                                    stack[SP] = PC;
                                    SP = SP + 4'B1;
                                    input_sel = 4'B0011;
                                    oper_decode = 4'B1011;
                                    dest_sel = 3'B010;
                                    PC_alt_cmnd_ex =  1;
```

```
                                    end

                        5'B11011: begin                    //HLT
stops processor
                                    run         = 1;
   //normally run = 0 here
                                    oper_decode  = 4'B1111;
                                    dest_sel     = 3'B111;
                                    start        = 1;    //for
normal operation remove this line
                                    finish       = 1;
                                  end
                        5'B11100: begin                    //set
start flag
                                    oper_decode = 4'B1111;
                                    dest_sel    = 3'B111;
                                    start       = 0;    //for
normal operation start = 1;
                                  end
                        5'B11101: begin                    //DEC
A<=A-1, B<=B-1
                                    if (IR1[10:9] == 2'B00)
                                      begin
                                        input_sel   =
4'B0001;
                                        dest_sel    = 3'B100;
                                      end
                                    else
                                      begin
                                        input_sel   = 4'B0010;
                                        dest_sel    = 3'B101;
                                      end
                                    oper_decode = 4'B0001;
                                  end
                        5'B11110: begin                    //INC
A<=A+1, B<=B+1
                                    if (IR1[10:9] == 2'B00)
                                      begin
                                        input_sel   =
4'B0001;
                                        dest_sel    = 3'B100;
                                      end
                                    else
                                      begin
                                        input_sel   =
4'B0010;
                                        dest_sel    = 3'B101;
                                      end
                                    oper_decode = 4'B0000;
                                  end

              endcase
              state                 = 4'B0101;
            end
//WAIT1       GIVES ALU AND BUSES TIME TO
UPDATE THEIR REGISTERS
//                  ALSO CONTAINS HALTING LOOP
                    4'B0101: begin
                                if (run)
                                  state = 4'B0110;
                                else
                                  state = 4'B0101;
                              end
//WAIT2
                    4'B0110: begin
                                  state = 4'B1000;
                              end
//EXECUTE
                    4'B1000: begin
                                  C       = temp_C;
                                  ram_data_storage =
temp_ram_data_storage;

                                  if (PC_alt_cmnd_ex)
                                  PC    = temp_PC;

                                  A       = temp_A;
                                  B       = temp_B;

                                  if (!flags_alt_cmd_ex)
                                    flags  = temp_flags;
                                  state    = 4'B1001;
                                end
//STORE
                    4'B1001: begin
                                  write_en  = 1;
                                  out_en    = 1;
                                  dest_sel  = 3'B111;
                                  state     = 4'B0000;
                                end

            endcase
   end
endmodule
```

into_alu_mux.v

```
module into_alu_mux (in1, in2, in3, in4, in5, in6, ch_sel,
out1, clk);
   input [15:0] in1;    //A
   input [15:0] in2;    //B
   input [31:0] in3;    //C
   input [15:0] in4;    //IR2
   input [15:0] in5;    //ram_data
   input [15:0] in6;    //PC
//  input [7:0]  in7;      //PortA
   input [3:0]  ch_sel;
   output [31:0] out1;

   input clk;

   reg [31:0] out1;

   always @(negedge clk)
   begin
    case (ch_sel)
      4'B0000: out1 = {in1, in2};
      4'B0001: out1 = {in1, in4};
      4'B0010: out1 = {in2, in4};
      4'B0011: out1 = {16'B0, in4};
      4'B0100: out1 = {16'B0, in6};
      4'B0101: out1 = {16'B0, in5};
      4'B0110: out1 = {16'B0, in1};
      4'B0111: out1 = {16'B0, in2};
//    4'B1000: out1 = {24'B0, in7};
      4'B1001: out1 = in3;
      default: out1 = 32'B0;
    endcase
   end
endmodule
```

out_alu_demux.v

```
module out_alu_demux(in1, ch_sel, out1, out2, out3, out5,
out6, clk);
```

```
    input [31:0] in1;
    input [2:0]  ch_sel;

    output [31:0] out1;  //C
    output [15:0] out2;  //ram_data
    output [15:0] out3;  //PC
//  output [7:0]  out4;  //PortA
    output [15:0] out5;  //A
    output [15:0] out6;  //B

    input clk;

    reg [31:0] out1;
    reg [15:0] out2, out3, out5, out6;

    always @(negedge clk)
    begin
      case (ch_sel)
          3'B000: begin
                    out1 = in1;
//                  out2 = out2;
//                  out3 = out3;
//                  out4 = out4;
//                  out5 = out5;
//                  out6 = out6;
                  end
          3'B001: begin
//                  out1 = out1;
                    out2 = in1;
//                  out3 = out3;
//                  out4 = out4;
//                  out5 = out5;
//                  out6 = out6;
                  end
          3'B010: begin
//                  out1 = out1;
//                  out2 = out2;
                    out3 = {16'B0, in1[15:0]};
//                  out4 = out4;
//                  out5 = out5;
//                  out6 = out6;
                  end
          3'B100: begin
//                  out1 = out1;
//                  out2 = out2;
//                  out3 = out3;
//                  out4 = out4;
                    out5 = {16'B0, in1[15:0]};
//                  out6 = out6;
                  end
          3'B101: begin
//                  out1 = out1;
//                  out2 = out2;
//                  out3 = out3;
//                  out4 = out4;
//                  out5 = out5;
                    out6 = {16'B0, in1[15:0]};
                  end
          3'B110: begin
//                  out1 = out1;
//                  out2 = out2;
//                  out3 = out3;
//                  out4 = out4;
                    out5 = in1[31:16];
                    out6 = in1[15:0];
                  end

          default: begin
                    out1 = out1;
                    out2 = out2;
                    out3 = out3;
//                  out4 = out4;
                    out5 = out5;
                    out6 = out6;
                  end
      endcase
    end
endmodule
```

y42_alu.v

```
module y42_alu (input1, input2, carry, oper_sel, output1,
Cflag, Zflag, Gflag, Lflag, clk);
    input [15:0]  input1;
    input [15:0]  input2;
    input         carry;
    input [3:0]   oper_sel;
    output[31:0]  output1;
    output       Cflag, Zflag, Gflag, Lflag;

    input clk;

    reg [31:0]    output1, output2;
    reg        Cflag, Zflag, Gflag, Lflag;

    integer i;

    always @(negedge clk)
    begin
      output1             = 32'B0;
      case (oper_sel)
        4'B0000: begin
                   output1 = input1 + input2;
                   Cflag   = input1[15] & input2[15];
                 end
        4'B0001: begin
                   output1 = input1 - input2;
//                 Zflag   = (input1 == input2);
                   Gflag   = (input1  > input2);
                   Lflag   = (input1  < input2);
                   Cflag   = (Lflag);
                 end
        4'B0010: begin
                    for (i = 0; i < 8; i= i + 1)
                      begin
                        output2 = input1 & {8{input2[i]}};
                        output1 = output1 + (output2 << i);
                      end
                 end
        4'B0100: output1 = {16'B0, input1 & input2};
        4'B0101: output1 = {16'B0, input1 | input2};
        4'B0110: output1 = {16'B0, ~input2};
        4'B0111: output1 = {16'B0, (~input1 & input2) |
(input1 & ~input2)};
//      4'B1000: Gflag = input1 > input2;
//      4'B1001: Lflag = input1 < input2;
//      4'B1010: Zflag = input1 == input2;
        4'B0011: output1 = {input1, input2} + carry;
        4'B1000: output1 = {input1, input2} - carry;
        4'B1011: output1 = {input1, input2};
        4'B1100: output1 = {16'B0, input1 & ~input2};
      //for BCLR
        4'B1101: output1 = input2 >> 1;     //Shift right 1
        4'B1110: output1 = input2 << 1;     //Shift left 1
        default: begin
                   output1 = 32'B1;
                   Zflag   = 0;
                 end
```

```
      endcase
      if (output1 == 32'B0)
        Zflag = 1;
      else
        Zflag = 0;
      if (oper_sel != 4'B0001)
         begin
            Lflag     = 0;
            Gflag     = 0;
         end
      Cflag = carry;
   end
endmodule
```

y42_processor.ucf

```
NET "address<0>" LOC = "p67";
NET "address<1>" LOC = "p66";
NET "address<2>" LOC = "p65";
NET "address<3>" LOC = "p64";
NET "address<4>" LOC = "p63";
NET "address<5>" LOC = "p57";
NET "address<6>" LOC = "p56";
NET "address<7>" LOC = "p55";
NET "address<8>" LOC = "p54";
NET "address<9>" LOC = "p53";
NET "address<10>" LOC = "p108";
NET "address<11>" LOC = "p107";
NET "address<12>" LOC = "p103";
NET "address<13>" LOC = "p102";
NET "address<14>" LOC = "p101";
NET "address<15>" LOC = "p100";
NET "address<16>" LOC = "p99";
NET "address<17>" LOC = "p97";
NET "address<18>" LOC = "p96";
NET "ram_data<0>" LOC = "p70";
NET "ram_data<1>" LOC = "p71";
NET "ram_data<2>" LOC = "p72";
NET "ram_data<3>" LOC = "p73";
NET "ram_data<4>" LOC = "p74";
NET "ram_data<5>" LOC = "p78";
NET "ram_data<6>" LOC = "p79";
NET "ram_data<7>" LOC = "p80";
NET "ram_data<8>" LOC = "p81";
NET "ram_data<9>" LOC = "p82";
NET "ram_data<10>" LOC = "p84";
NET "ram_data<11>" LOC = "p85";
NET "ram_data<12>" LOC = "p86";
NET "ram_data<13>" LOC = "p87";
NET "ram_data<14>" LOC = "p93";
NET "ram_data<15>" LOC = "p94";
NET "write_en" LOC = "p68";
NET "chip_sel" LOC = "p109";
NET "out_en" LOC = "p95";
NET "clk" LOC = "p89";
NET "rst" LOC = "p161";
NET "start" LOC = "p200";
NET "finish" LOC = "p199";
```

Note: Because the program that loads software into the external RAM on the XSV board counts the number of bytes rather than the number of words, a variable stored at at address 0x0100 will appear in a memory dump at address 0x0200. All binary files are written in XESS-16 format.

hardware_mult.XES

ASSEMBLY:

```
START
LDA  #$0007
LDB  #$0006
MUL
STAC RESULT
FINISH
```

BINARY:

- 10 0000 E0 00 00 00 5A 00 00 07 62 00 00 06 18 00 00 00
- 10 0010 6C 00 01 05 D8 00 00 00 00 00 00 00 00 00 00 00
- 10 0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0040 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF
- 10 0050 FF FF FB FF FF BF FF FF FD FF FF FF FB FF DE F7
- 10 0060 EF FF FF FF FF FF FF FF DF FF EF EF FF FD FF EF
- 10 0070 FF FF FF BF FF FF FF FF FF DF DF 7D FF FF FF FF
- 10 0080 DA FF FF BF FF FF FF FF FF FF FF FF 7F FF DF FF
- 10 0090 FF 7F FF FF FF FF FF FF F7 FF DF FD FD FF BE FF
- 10 00A0 FF FF FD FF FF FF FF B7 FB FF FF FF F7 FF FF FF
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF F7 FD
- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF FF FF
- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF F7 FF
- 10 0170 FF FF FF FF FF FF FF FF FF BF FF FF FF FF FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF EF FF
- 10 01F0 EF FF FF FF FF FF 7F F7 FF FF FF FF FF FF FD FD
- 10 0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

software_mult.XES

ASSEMBLY:

```
START
```

```
    LDA #$0000
    STA   SUM
    STA BIT_NUM
    STA CNT
    LDA  #$0007
    LDB #$0006
    STA   X
    STB Y
  LOOP:
    LDB   Y
    LDA  #$0001
    AND
    BEQ   SKIP_BIT
    LDA  X
    LDB  #$00FF
    AND
    MOVCB
    LDA  BIT_NUM
  SHIFT_SUM:
    CMPA   #$0000
    BEQ   SHIFT_DONE
    SHLB
    MOVCB
    DECA
    JMP   SHIFT_SUM
  SHIFT_DONE:
    LDA  SUM
    ADD
    STAC SUM
  SKIP_BIT:
    LDA  Y
    SHRA
    STAC Y
    LDA  BIT_NUM
    INCA
    STA   BIT_NUM
    CMPA   #$0008
    BNE   LOOP
    STAC RESULT
    FINISH
```

BINARY:

```
- 10 0000 E0 00 00 00 5A 00 00 00 68 00 01 00 68 00 01
01
- 10 0010 68 00 01 02 5A 00 00 07 62 00 00 06 68 00 01
03
- 10 0020 6A 00 01 04 60 00 01 04 5A 00 00 01 28 00 00
00
- 10 0030 80 00 00 1B 58 00 01 03 62 00 00 FF 28 00 00
00
- 10 0040 72 00 00 00 58 00 01 01 4A 00 00 00 80 00 00
18
- 10 0050 C2 00 00 00 72 00 00 00 E8 00 00 01 78 00 00
12
- 10 0060 58 00 01 00 08 00 00 00 6C 00 01 00 58 00 01
04
- 10 0070 C8 00 00 00 6C 00 01 04 58 00 01 01 F0 00 00
01
- 10 0080 68 00 01 01 4A 00 00 04 88 00 00 09 6C 00 01
05
- 10 0090 D8 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00
- 10 00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF
F7 FD
- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF
FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF
FF FF
- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB
FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF
FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF
FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF
FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF
CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF
FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF
F7 FF
- 10 0170 FF FF FF FF FF FF FF FF FF BF FF FF FF FF
FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF
FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF
FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF
FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF
FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF
F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF
FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF
EF FF
- 10 01F0 EF FF FF FF FF FF FF 7F F7 FF FF FF FF FF FF
FD FD
- 10 0200 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF
```

hardware_xor.XES

ASSEMBLY:

```
  START
  LDA   #$FF00
  LDB #$FFFF
  XOR
  STAC RESULT
  FINISH
```

BINARY:

```
- 10 0000 E0 00 00 00 5A 00 FF 00 62 00 FF FF 40 00 00
00
- 10 0010 6C 00 01 03 D8 00 00 00 00 00 00 00 00 00 00
00
- 10 0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0050 FF FF FB FF FF BF FF FF FD FF FF FF FB FF
DE F7
- 10 0060 EF FF FF FF FF FF FF FF DF FF EF EF FF FD
FF EF
- 10 0070 FF FF FF BF FF FF FF FF FF DF DF 7D FF FF
FF FF
- 10 0080 DA FF FF BF FF FF FF FF FF FF FF FF 7F FF
DF FF
- 10 0090 FF 7F FF FF FF FF FF FF F7 FF DF FD FD FF
BE FF
- 10 00A0 FF FF FD FF FF FF FF B7 FB FF FF FF F7 FF
FF FF
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF
F7 FD
```

- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF FF FF
- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF F7 FF
- 10 0170 FF FF FF FF FF FF FF FF BF FF FF FF FF FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF EF FF
- 10 01F0 EF FF FF FF FF FF 7F F7 FF FF FF FF FF FF FD FD
- 10 0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

software_xor.XES

ASSEMBLY:

```
START
LDA
STA   X
NOTA
MOVCA
AND
STAC SUM1
LDA   X
LDB   Y
NOTB
MOVCB
AND
MOVCB
LDA   SUM1
OR
STAC RESULT
FINISH
```

BINARY:

- 10 0000 E0 00 00 00 5A 00 FF 00 68 00 01 00 62 00 FF FF
- 10 0010 6A 00 01 01 38 00 00 00 70 00 00 00 28 00 00 00
- 10 0020 6C 00 01 02 58 00 01 00 60 00 01 01 3A 00 00 00
- 10 0030 72 00 00 00 28 00 00 00 72 00 00 00 58 00 01 02
- 10 0040 30 00 00 00 6C 00 01 03 D8 00 00 00 FF FF FF FF
- 10 0050 FF FF FB FF FF BF FF FF FD FF FF FF FB FF DE F7
- 10 0060 EF FF FF FF FF FF FF FF DF FF EF EF FF FD FF EF
- 10 0070 FF FF FF BF FF FF FF FF FF DF DF 7D FF FF FF FF
- 10 0080 DA FF FF BF FF FF FF FF FF FF FF FF 7F FF DF FF
- 10 0090 FF 7F FF FF FF FF FF FF F7 FF DF FD FD FF BE FF
- 10 00A0 FF FF FD FF FF FF FF B7 FB FF FF FF F7 FF FF FF
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF F7 FD
- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF FF FF
- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF F7 FF
- 10 0170 FF FF FF FF FF FF FF FF FF BF FF FF FF FF FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF EF FF
- 10 01F0 EF FF FF FF FF FF 7F F7 FF FF FF FF FF FF FD FD
- 10 0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

hardware_multT.XES

ASSEMBLY:

```
LOOP:
   START
   LDA   #$0007
   LDB   #$0006
   MUL
   STAC RESULT
   FINISH
   JMP LOOP
```

BINARY:

```
- 10 0000 E0 00 00 00 5A 00 00 07 62 00 00 06 18 00 00
00
- 10 0010 6C 00 01 05 D8 00 00 00 78 00 00 00 00 00 00
00
- 10 0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0040 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF
FF
- 10 0050 FF FF FB FF FF BF FF FF FD FF FF FF FB FF
DE F7
- 10 0060 EF FF FF FF FF FF FF FF DF FF EF EF FF FD
FF EF
- 10 0070 FF FF FF BF FF FF FF FF FF DF DF 7D FF FF
FF FF
- 10 0080 DA FF FF BF FF FF FF FF FF FF FF FF 7F FF
DF FF
- 10 0090 FF 7F FF FF FF FF FF FF F7 FF DF FD FD FF
BE FF
- 10 00A0 FF FF FD FF FF FF FF B7 FB FF FF FF F7 FF
FF FF
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF
F7 FD
- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF
FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF
FF FF
- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB
FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF
FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF
FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF
FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF
CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF
FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF
F7 FF
- 10 0170 FF FF FF FF FF FF FF FF FF BF FF FF FF FF
FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF
FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF
FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF
FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF
FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF
F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF
FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF
EF FF
- 10 01F0 EF FF FF FF FF FF 7F F7 FF FF FF FF FF FF
FD FD
- 10 0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

software_multT.XES

ASSEMBLY:

```
    OUTER_LOOP:
      START
      LDA #$0000
      STA   SUM
      STA BIT_NUM
      STA CNT
      LDA  #$0007
      LDB #$0006
      STA   X
      STB Y
    LOOP:
      LDB   Y
      LDA  #$0001
      AND
      BEQ   SKIP_BIT
      LDA   X
      LDB  #$00FF
      AND
      MOVCB
      LDA  BIT_NUM
    SHIFT_SUM:
      CMPA   #$0000
      BEQ   SHIFT_DONE
      SHLB
      MOVCB
      DECA
      JMP   SHIFT_SUM
    SHIFT_DONE:
      LDA  SUM
      ADD
      STAC SUM
    SKIP_BIT:
      LDA  Y
      SHRA
      STAC Y
      LDA  BIT_NUM
      INCA
      STA   BIT_NUM
      CMPA   #$0008
      BNE   LOOP
      STAC RESULT
      FINISH
      JMP OUTER_LOOP
```

BINARY:

```
- 10 0000 E0 00 00 00 5A 00 00 00 68 00 01 00 68 00 01
01
- 10 0010 68 00 01 02 5A 00 00 07 62 00 00 00 68 00 01
03
- 10 0020 6A 00 01 04 60 00 01 04 5A 00 00 01 28 00 00
00
- 10 0030 80 00 00 1B 58 00 01 03 62 00 00 FF 28 00 00
00
- 10 0040 72 00 00 00 58 00 01 01 4A 00 00 00 80 00 00
18
- 10 0050 C2 00 00 00 72 00 00 00 E8 00 00 01 78 00 00
12
- 10 0060 58 00 01 00 08 00 00 00 6C 00 01 00 58 00 01
04
- 10 0070 C8 00 00 00 6C 00 01 04 58 00 01 01 F0 00 00
01
- 10 0080 68 00 01 01 4A 00 00 04 88 00 00 09 6C 00 01
05
- 10 0090 D8 00 00 00 78 00 00 00 00 00 00 00 00 00 00
00
- 10 00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF
F7 FD
- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF
FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF
FF FF
```

- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF F7 FF
- 10 0170 FF FF FF FF FF FF FF FF FF BF FF FF FF FF FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF EF FF
- 10 01F0 EF FF FF FF FF FF FF 7F F7 FF FF FF FF FF FF FD FD
- 10 0200 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

hardware_xorT.XES

ASSEMBLY:

```
LOOP:
   START
   LDA  #$FF00
   LDB #$FFFF
   XOR
   STAC RESULT
   FINISH
   JMP LOOP
```

BINARY:

- 10 0000 E0 00 00 00 5A 00 FF 00 62 00 FF FF 40 00 00 00
- 10 0010 6C 00 01 03 D8 00 00 00 78 00 00 00 00 00 00 00
- 10 0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0050 FF FF FB FF FF BF FF FF FD FF FF FF FB FF DE F7
- 10 0060 EF FF FF FF FF FF FF FF DF FF EF EF FF FD FF EF
- 10 0070 FF FF FF BF FF FF FF FF FF DF DF 7D FF FF FF FF
- 10 0080 DA FF FF BF FF FF FF FF FF FF FF FF 7F FF DF FF
- 10 0090 FF 7F FF FF FF FF FF FF F7 FF DF FD FD FF BE FF
- 10 00A0 FF FF FD FF FF FF FF B7 FB FF FF FF F7 FF FF FF
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF F7 FD
- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF FF FF
- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF F7 FF
- 10 0170 FF FF FF FF FF FF FF FF FF BF FF FF FF FF FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF EF FF
- 10 01F0 EF FF FF FF FF FF FF 7F F7 FF FF FF FF FF FF FD FD
- 10 0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

software_xorT.XES

ASSEMBLY:

```
LOOP:
   START
   LDA
   STA   X
   NOTA
   MOVCA
   AND
   STAC SUM1
   LDA   X
   LDB   Y
   NOTB
   MOVCB
   AND
   MOVCB
   LDA   SUM1
   OR
   STAC RESULT
   FINISH
   JMP LOOP
```

BINARY:

- 10 0000 E0 00 00 00 5A 00 FF 00 68 00 01 00 62 00 FF FF

- 10 0010 6A 00 01 01 38 00 00 00 70 00 00 00 28 00 00 00
- 10 0020 6C 00 01 02 58 00 01 00 60 00 01 01 3A 00 00 00
- 10 0030 72 00 00 00 28 00 00 00 72 00 00 00 58 00 01 02
- 10 0040 30 00 00 00 6C 00 01 03 D8 00 00 00 78 00 00 00
- 10 0050 FF FF FB FF FF BF FF FF FD FF FF FF FB FF DE F7
- 10 0060 EF FF FF FF FF FF FF FF DF FF EF EF FF FD FF EF
- 10 0070 FF FF FF BF FF FF FF FF FF DF DF 7D FF FF FF FF
- 10 0080 DA FF FF BF FF FF FF FF FF FF FF FF 7F FF DF FF
- 10 0090 FF 7F FF FF FF FF FF FF F7 FF DF FD FD FF BE FF
- 10 00A0 FF FF FD FF FF FF FF B7 FB FF FF FF F7 FF FF FF
- 10 00B0 7F FB FF FB FF FF FD FF FE F7 FF FF FF FF F7 FD
- 10 00C0 FF FF FF FF FE F7 FF BF FF FF FF FF FF FF FA 7F
- 10 00D0 EF FF FF FF FF FF FF FF FE FF 7F FB FF BF FF FF
- 10 00E0 FF FF FF FD EF FF FF FF FF F7 F7 FF FF FB FF FF
- 10 00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FB
- 10 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 10 0110 FF BF FD A7 FF DE FF FF FF FF FB FF FF FF FF FF
- 10 0120 FF FF FF BF FF FF FF FF FF FF 7F EF FF FF FF FE
- 10 0130 FF EF FD FF FF DF FF FF DF FF FF FF FD FF FF FF
- 10 0140 FF FF FF FD FF FF FF FF FF FF FF FF FF FF CF FF
- 10 0150 FE FF FF FF FF BF BF EF FF FF FD F7 7F FF FD FF
- 10 0160 FF FF FF FF FD FF FF FF FF FF FF 7F 7F FF F7 FF
- 10 0170 FF FF FF FF FF FF FF FF FF BF FF FF FF FF FE FF
- 10 0180 FF FF FF FF FF FF FF FF FF FF FF FF BF FF FF FF
- 10 0190 FF FF FF FF FF FF FF FF FF FF FF EF FB FF FD FF
- 10 01A0 FF FF FF FF FF FF FF FF FF FB EF FF FF FF FF DF
- 10 01B0 1F FF FF FD EF FF FF FB FF FF FB FE FF FF FF FE
- 10 01C0 FF DF FF FF BF BF FE FF FF FB FD FE FF FF F7 FF
- 10 01D0 FF FF 7F FF FF FF FB FF FF FF FF F7 FF FF FF FF
- 10 01E0 FB FF FF FF ED FF FF FF FF EF DF FF FF FF EF FF
- 10 01F0 EF FF FF FF FF FF 7F F7 FF FF FF FF FF FF FD FD
- 10 0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Instruction Set Summary

MNEMONIC	**OP-CODE** (IR1bit 15…IR1bit11)	**OPTIONS** (IR1bit10…IR1bit9)	**OPERATION**
ADD	00001	XX	C <= A+B
SUB	00010	XX	C <= A-B
MUL	00011	XX	C<=A*B
RTS	00100	XX	PC <= STK[SP]
AND	00101	XX	C <= A & B
OR	00110	XX	C <= A | B
NOT	00111	00	C = ~A
		01	C = ~B
EXOR	01000	XX	C <= A^B;
CMP	01001	00	FLAGS <= A-B
		01	FLAGS <= A-IR2
		10	FLAGS <= B-IR2
LDA	01011	00	A <= [M]
		01	A <= IR2
LDB	01100	00	B <= [M]
		01	B <= IR2
STAx	01101	00	IR2[M] <= A
		01	IR2[M] <= B

		10	IR2[M] <= LoC
MOVCx	01110	00	A <= LoC
		01	B <= LoC
		10	A:B <= C
JMP	01111	XX	PC <= k
BEQ	10000	XX	Z=1? PC <= IR2
BNE	10001	XX	Z=0? PC <= IR2
BGE	10010	XX	G=1? PC <=IR2
BLT	10011	XX	L=1? PC <=IR2
BSET	10100	00	C<=IR2|A
		01	C<=IR2|B
BCLR	10101	00	C<=~IR2&A
		01	C<=~IR2&B
CLC	10110		CARRY <= 0
HLT	11011		STSB <= 1
SEC	10111		CARRY <= 1
SHRx	11001	00	C <= A>>1
		01	C <= B>>1
SHLx	11000	00	C <= A<<1
		01	C<= B<<1
INCx	11110	00	A <= A+IR2
		01	B <= B+IR2
DECx	11101	00	A <= A-IR2
		01	B <= B-IR2
START	11100	XX	STSB <= 0
JSR	11010	XX	STK[SP] <=PC,PC<=IR2