

14:332:437 Digital Systems Design
Fall 2007
Problem Set 4
Electrical and Computer Engineering Department
College of Engineering
Rutgers University

Prof. Michael L. Bushnell

Assigned: October 17, 2007

Due: October 24, 2007

STUDENTS ARE EXPECTED TO WORK INDEPENDENTLY ON BOTH HOMEWORK AND EXAMINATIONS. NO COLLABORATION IS PERMITTED, UNLESS YOU ARE OTHERWISE INSTRUCTED. YOUR WORK MUST BE YOUR OWN.

1. *Verilog Microprocessor Design.*

Introduction. The following circuit is a big-end microprocessor (similar to an ARMCORE) data path with some of the control logic. Please describe behaviorally in VERILOG what the hardware in Figure 1 does. There should be separate modules for the *RegisterFile*, *ALU*, and *PC*. Module *control* is a finite state machine that sequences all of the control lines in the figure for the other modules.

Instruction Format: This machine is a 3-address machine. The *OPCODE* field of *IR* says what instruction to do, *R1* of *IR* says what register is gated onto the *ABUS*, *R2* of *IR* says what register is gated onto the *BBUS*, and *R3* of *IR* says where to store the result on the *CBUS*.

Register File Operation: You operate the *RegisterFile* by wiring *R1* to *ABUSread*, *R2* to *BBUSread*, and *R3* to *CBUSwriteMSB*. *CBUSwriteMSB* gives the register that the 32 MSBs of the *CBUS* are written to, and *CBUSwriteLSB* gives the register that the 32 LSBs of the *CBUS* are written to. The memory is not written from the *CBUS* unless signal *activateLSB* or *activateMSB* is asserted to 1. There are 32 general purpose registers, of 32 bits, named *X0* through *X31*.

Instruction Descriptions: In order to do arithmetic instructions, you must assert one and only one of the 3 control line sets (*subtract*, *{right shift, arithmetic}*, or *{double left shift, arithmetic}*)

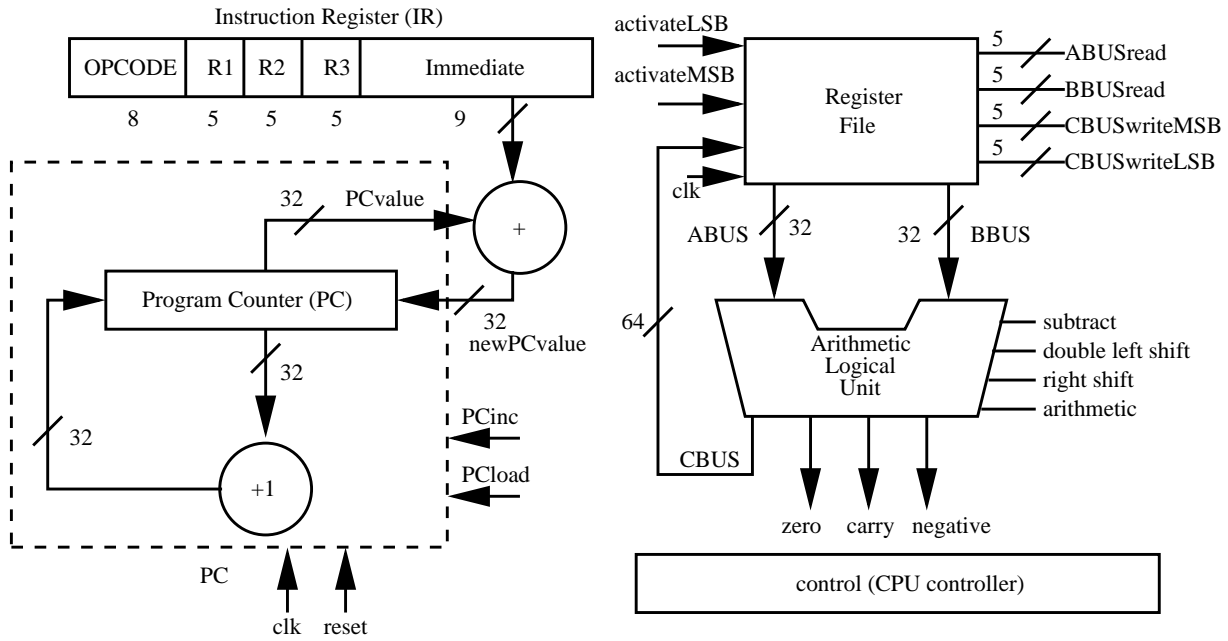


Figure 1: Microprocessor block diagram.

to tell the *ALU* what to do. The *ALU* operates on the *ABUS* and *BBUS* and puts its result on the *CBUS*. For *subtract* ($(R1) - (R2)$) and *arithmetic right shift*, only the 32 least significant bits are set on the *CBUS*. The *arithmetic right shift* replicates the sign bit (bit 0) to fill in bits vacated as the machine word shifts right. The last bit coming out of the least significant bit position of the shifted word should be kept in the *carry* signal. To activate *arithmetic right shift*, set both the *right shift* and *arithmetic* signals. For *double left shift*, *ABUS* is shifted left by *BBUS* giving a 64-bit result, which is put on the entire 64-bit *CBUS*. Both the *arithmetic* and *double left shift* signals must be simultaneously asserted for this to work. For the *double left shift* instruction only, please store the 32 most significant bits (of *CBUS*) in *R3* and the 32 least significant bits (of *CBUS*) in *R3 + 1*.

Instructions to Implement: You are now ready to implement the following instructions with the specified hexadecimal *OPCODEs*. Assume that some other hardware has already pre-fetched the instruction to be executed into the *Instruction Register*. Don't forget to add 1 to the *PC* after the instruction executes, so that the machine will fetch the next sequential instruction. Branches are executed by adding the *Immediate* field of *IR* to the *PC* and storing the result in the *PC*, as follows: $PC \leftarrow Immediate + (PC)$.

OPCODE	Operation	Meaning
8'h3A	<i>subtract</i>	$R3 \leftarrow (R1) - (R2)$
8'h21	<i>right shift</i>	$R3 \leftarrow (R1) \gg (R2)$
8'h22	<i>double left shift</i>	$\{R3, R3 + 1\} \leftarrow (R1) \ll (R2)$
8'h14	<i>branch if zero</i>	$PC \leftarrow (PC) + Immediate$ only if <i>zero</i> is set

Instruction Timing: In the first clock after the machine enters the **RESET** state of the CPU controller, assume that an instruction has been fetched, and operate the *ALU* to execute that instruction and produce its result on the *CBUS* (including updating the *PC*). If the

instruction is *branch*, then go to the RESET state at the second clock. Otherwise, in the second clock, store the *CBUS* back into the *RegisterFile*, and then return to the RESET state at the third clock. In addition to all signals in Figure 1, the machine has active high signals *clk* and *reset*, which puts the CPU controller in the RESET state and clears the *PC*.

Work to be done:

- (a) Write complete VERILOG code for the modules *RegisterFile*, *ALU*, and *PC*.
- (b) Declare the *Instruction Register (IR)* as a 32-bit input port in module *control*. This port contains the last instruction fetched for execution. Declare and set the sub-fields of *IR* as wires to make your VERILOG code more readable. Write assign statements that extract the bits from the 5 fields of *IR* and store them on wires named after those fields.
- (c) Write an *always* block for the *control* module next state decoder that cycles the machine from the RESET state through execution of the 4 instructions. This block should be triggered by any change in the *present_state* of the controller. Also, write the *always* block to update *present_state* from the *next_state*.
- (d) After each *subtract*, *double left shift*, or *arithmetic right shift* operation, set 3 registers (*carry*, *zero*, or *negative*) in the processor to indicate the status of the result (for *double left shift*, give the status of the entire 64-bit result). Do not change these registers for a *branch* operation. For a shift operation, set *zero* if the result register is zero, set *negative* if the MSB of the result register is a 1 bit, and set *carry* to the last bit shifted out of the result register.
- (e) Turn in a block diagram of the hardware (from Synopsys), your behavioral Verilog code, the optimized logic diagram of the hardware (from Synopsys), and a VCS simulation done using the test bench on the web site.

Note: This is the classic implementation of the VonNeumann three-address stored program computer architecture, and it is embodied in every machine made today (IBM System/360 (IBM ES9000), IBM PowerPC, Intel Pentium IV, Intel Itanium II, AMD Athelon, HP Precision Architecture, and Sun SPARC).