cs281: Introduction to Computer Systems

CPUlab – Datapath

Assigned: Oct. 29, Due: Nov. 3

The objective of this exercise is to familiarize you with the Datapath of the Y86 CPU and to introduce you to the fundamental concepts required for control of the execution of instructions in a sequential implementation of the CPU. Functional elements in the Datapath include:

PC : Program Counter – stateful device containing a 32 bit register holding the current program counter. Input is the upcoming PC value, output is the current PC value. A new program counter is stored on the rising clock edge.

IMem : Instruction Memory – stateful device containing the Y86 program, starting at address 0. Input is the 32 bit Program Counter, and output are the 6 consecutive bytes from memory starting at the PC address. Internally, memory is loaded into four separate banks, each of which is 2 bytes wide, and loading must occur prior to instruction execution. From that point on, the memory may be thought of as combinational, where, based on the PC as input, a six byte output is produced.

ISplit : Instruction Split – combinational circuit whose inputs are the 6 (potential) bytes of the instruction and whose outputs are the union of the fields that make up all the Y86 instructions, including icode, ifun, rA, rB, Dest, and V/D.

Registers : Register File – stateful device containing the eight Y86 registers, numbered from 0 to 7. The register file is capable of routing to output the current values of *two* of its registers, as well as updating one or two registers. Inputs are srcA and srcB, to determine values sent to the valA and valB outputs, as well as dstE/valE and dstM/valM, which determine register (and its associated value) to write on the next rising clock. If dstE and/or dstM have value 0xF, then no register is updated. Outputs valA and valB always reflect the current value of registers srcA and srcB.

ALU : Arithmetic Logic Unit – combinational circuit capable of performing addl, subl, andl, and xorl. Inputs are ALUfun, ALU_A, and ALU_B. Outputs are ZF, SF, OF, and valE.

CC : Condition Codes – stateful device implemented as a register holding the 3 1-bit values corresponding to ZF, SF, and OF. In addition to these inputs, the register has an enable bit and an asynchronous reset bit as inputs. The output is the current values of the three bits. Updates occur on the rising clock if the enable is also asserted.

DAddr : Data Memory Address – combinational circuit that, given a 32 bit aligned byte address yields a 20 bit word address suitable for input to the data memory. The device also has a single bit indicating whether or not an illegal address was given as its input.

DMem : Data Memory – stateful device used for the read/write data memory involved in a Y86 program. Input is the 20 bit word address A to be read or written along with the 32 bit data value, D, to be modified as a result of a write. Inputs also include a "Store" bit which, when 1, will cause the D value to be stored on the next rising clock edge, as well as an asynchronous reset for the memory. Output is the 32-bit valM corresponding to the value in memory at the word address given by A.

In addition to these functional elements, the datapath consists of a set of multiplexers, one adder unit, and the set of wires connecting all of the elements. The datapath can be made to "execute" instructions by manipulating the control bits, which in this initial datapath are simply input pins. By setting all of these control inputs to particular values based on the current instruction and on other outputs of the functional elements (like the condition codes), we can manipulate the datapath into performing the semantics of our given instructions.

The Table below gives the meanings for each of the control signals on the Y86 datapath.

Control	**Width**	**Description**
PCIncSrc	2	Determines value to add to PC to get to next instruction. 00 - 1, 01 - 2, 10 - 5, 11 - 6.
valCsrc	1	Determine value for valC. 0 means valC ←Dest (M4[PC+1]), 1 means valC ←V/D (M4[PC+2]).
valAsrc	1	Determine read register file output for valA. 0 means valA ←R[rA], 1 means valA ←R[%esp].
valBsrc	1	Determine read register file output for valB. 0 means valB ←R[rB], 1 means valB ←R[%esp].
dstEsrc	2	Determine destination register for write of valE. 00 means R[rB] ←valE, 01 means R[%esp] ←valE, 10 and 11 mean send 0xf to dstE, indicating no write.
dstMsrc	1	Determine destination register for write of valM. 0 means R[rA] ←valM, 1 means send 0xf to dstM, indicating no write.
aluAsrc	2	Determine value to send to ALU_A. 00 means ALU_A ←valA, 01 means ALU_A ←valC, 10 means ALU_A ←4, 11 means ALU_A ←-4.
aluBsrc	1	Determine value to send to ALU_B. 0 means ALU_B ←valB, 1 means ALU_B ←0.
setCC	1	Determine whether or not to update the CC register on the next clock. 0 indicates do not update, 1 indicates update.
aluOp	1	Determine operation to route to ALUfun. 0 means 0000 (add), 1 means use ifun.
dmemAddr	1	Determine address routed to data memory address line. 0 means dAddr ←valE. 1 means dAddr ←valA.
dmemData	1	Determine value routed to data memory D input. 0 means D ←valA, 1 means D ←valP
dmemWrite	1	Determine whether or not to store D at M4[dAddr] on the next clock. 0 indicates do not write memory, 1 indicates write memory.
newPC	2	Determine source of next Program Counter to be routed to input of PC register. 00 means newPC ←valP, 01 means newPC ←valC, 10 means newPC ←valM, and 11 is undefined.

Assignment

Your task in this assignment is twofold:

1. Learn the datapath by manually executing instructions in the datapath. A few initial instructions are given in the files example.ys and example.yo and their associated Logisim instruction memory image exbank0.mem, exbank1.mem, exbank2.mem, and exbank3.mem. Your professor will lead you through the first couple and general instructions are given below.

2. Your second task is to complete the table at the end of this handout completely defining the control functionality for all the given instructions. Based on the semantics of the instructions, you should determine the correct control signals for that instruction. The instruction semantics are given through the phases of the datapath in the textbook figures 4.18 through 4.21. You should use both pencil and paper as well as Logisim experimental techniques to determine these values.

Steps to execute one or more instructions:

1. Begin at the Datapath circuit on the canvas with the pointer tool selected. Open y86.circ and double-click the Datapath circuit in the explorer pane to make it the active circuit if this is not the case.

2. Right-click on the IMem device and select the 'View Memory' option.

3. For each of the memory banks, from top to bottom, you should right-click on the memory device and select the 'Load Image...' option.

4. Navigate in the file system and select the memory image appropriate to the particular bank (0, 1, 2, or 3).

5. Double-click on the Datapath circuit again to make it the active circuit.

6. Select the 'hand' tool so that we can manipulate values on the Datapath.

7. Take the clock to an asserted level.

8. Assert and then deassert the Reset pin. At this point, the PC, CC, and registers are initialized to zero and you should be able to see the values derivative of the first instruction at the probe displays (like icode, ifun, rA, and rB).

9. Instruction execution occurs in a single clock cycle. Once the instruction is available from IMem following the rising edge of the clock, we need to, based on the particular instruction, set the pins associated with the control signals in the datapath. Eventually, this manual operation will be replaced by a combinational circuit that performs the same task. For now, with the clock still high, set the control signals PCIncSrc, valCsrc, valAsrc, valBsrc, dstEsrc, dstMsrc, aluAsrc, aluBsrc, setCC, aluOp, dmemAddr, dmemData, dmemWrite,

and newPC to the correct values based on the instruction being executed. The professor will take you through the examples of irmovl and addl.

For irmovl, the values to be set include PCIncSrc=11, valCsrc=1, dstEsrc=00, dstMsrc=1, aluAsrc=01, aluBsrc=1, setCC=0, aluOp=0, dmemWrite=0, and newPC=00.

For addl, the values to be set include PCIncSrc=01, valAsrc=0, valBsrc=0, dstEsrc=00, dstMsrc=1, aluAsrc=00, aluBsrc=0, setCC=1, aluOp=1, dmemWrite=0, and newPC=00.

10. Once the control signals have been set, check the values from left to right through the datapath. Based on the register transfer semantics of the various instructions given in Tables 4.18 through 4.21 in your book, do the values of valP, valC, valA, valB, etc. correspond to the specific instruction being executed? If they do not, then verify your control signals again.

11. When the values/control signals are correct, change the clock to come down to the de-asserted state. Note that when the clock goes low, the negation in the lower left of the datapath will cause the clock input to the register file, the CC register, the data memory, and the PChold register to go high. So by bringing the clock low, these devices will take any write-values on their input and store them into their device. This, in effect, "seals the deal" for the execution of this instruction and we are ready to go on to the next instruction.

12. Change the clock back to the high/asserted state. The value of newPC is now stored in PC and we go back to step 9 and set the control signals appropriately.

Instruction	**PCIncSrc**	**valCsrc**	**valAsrc**	**valBsrc**	**dstEsrc**	**dstMsrc**	**aluAsrc**	**aluBsrc**	**setCC**	**aluOp**	**dmemAddr**	**dmemData**	**dmemWrite**	**newPC**
OPl rA,rB														
rrmovl rA,rB														
irmovl V,rB														
rmmovl rA,D(rB)														
mrmovl D(rB),rA														
pushl rA														
popl rA														
jXX Dest														
cmovXX rA, rB														
call Dest														
ret														
nop														
halt														