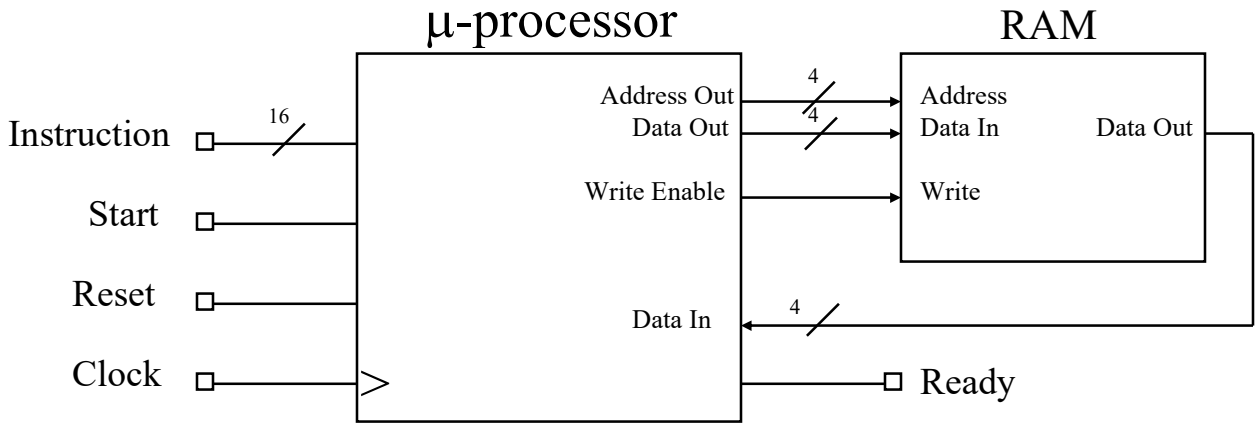
Lab 11/12: A Simple Microprocessor: The DDmini

PURPOSE

The purpose of this lab is to design and build - using the laboratory simulator - a simple multiple-cycle microprocessor with a micro-programmed control using implicit sequencing. Although implementing a full computer system is beyond scope of this course, we can implement the core of a simple microprocessor. This microprocessor lacks an Instruction Pointer / Program Counter and instruction memory for a stored program – the student fills this role during testing. Implementing the Digital Design mini (DDmini) processor is a two-week final project.

STRUCTURAL OVERVIEW

The DDmini is essentially a 4-bit microprocessor that uses at 16x4 RAM to store data. DDmini instructions are 16 bits wide and have 4-bit opcodes. All data is stored in two's complement representation. There is no separate support for unsigned magnitude binary numbers.



The DDmini microprocessor has the inputs and outputs shown above and described below:

CLOCK:	The external clock signal for the synchronous system.
RESET:	A 1-bit value that resets the computer to its idle/ready state when asserted.
READY:	A 1-bit output that is asserted high only when the microprocessor is in the idle state and awaiting a new instruction.
START:	A 1-bit value that must be asserted for one clock tick to instruct the computer to execute a new INSTRUCTION.
INSTRUCTION:	A 16-bit value representing a computer operation. The value of INSTRUCTION must be held stable from the beginning of its execution (the assertion of START) to the end of its execution (the assertion of READY).
ADDRESS OUT:	A 4-bit output that provides the binary address of a desired memory word.
DATA IN:	A 4-bit input that accepts a data word from memory.
DATA OUT:	A 4-bit output that provides a data word to be stored in memory.
WRITE ENABLE:	A 1-bit output that enables a memory write.

BEHAVIORAL OVERVIEW

The DDmini does not have a memory area for instructions nor does it have a program counter. Instead, that role is played by the student (or device testbed). This microprocessor uses the signal READY to request its next instruction and commences execution of an instruction when it receives the signal START.

Upon initialization (via the RESET input) the DDmini asserts READY and awaits its first instruction. Once the instruction is stable, the external system must assert START to indicate that the instruction should be executed. READY is not asserted again until the microprocessor completes the specified operation. Once the instruction is finished, the microprocessor asserts READY and awaits its next instruction.

This simple computer has a very small instruction set. Furthermore, the instructions are very basic; the microprocessor does not have general purpose registers that are addressable in its instruction set. Instead, all data is kept in a small RAM. In other words, the DDmini is a *memory-to-memory* processor.

The 16-bit instruction is composed of four 4-bit fields:

15:12	11:8	7:4	3:0
OPCODE	OPERAND A	OPERAND B	OPERAND C

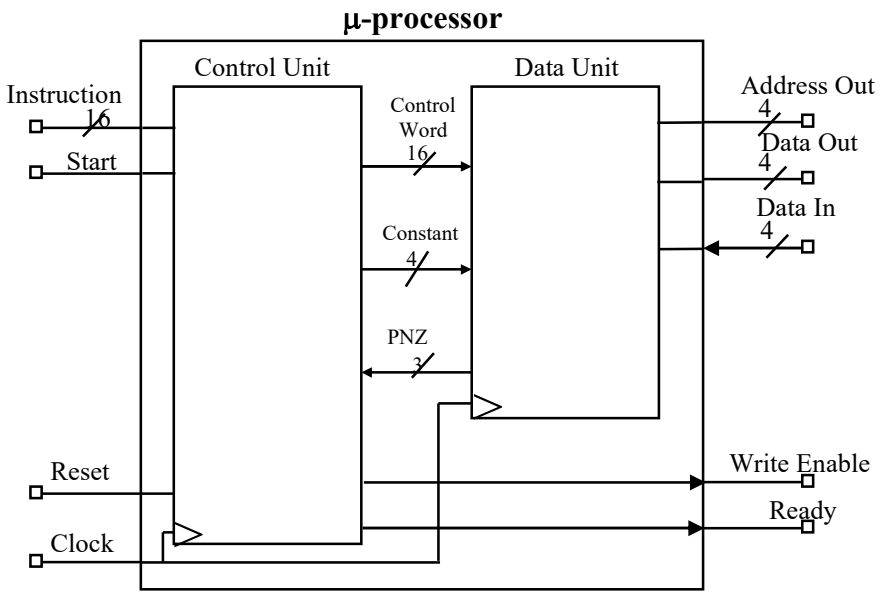
The OPCODE field defines the type of operation to be performed. The OPERAND values provide the data that exactly specifies the operation. The microprocessor supports the following instructions:

OPCODE	INSTRUCTION FUNCTION (RTL)
x0	NOP/IDLE
x1	M[A] ← B
x2	M[A] ← M[B]
x3	M[A] ← M[B] + M[C]
x4	M[A] ← M[B] - M[C]
x5	M[A] ← M[B] * M[C]
x6	M[A] ← M[B] >> 1
x7	M[A] ← NOT (M[B])
x8	M[A] ← M[B] AND M[C]
x9	IF (M[C] = 0) THEN M[A] ← M[B]
xA	IF (M[C] = M[B]) THEN (M[A] ← 0) ELSE (M[A] ← 1)
xB-xF	Reserved for TA madness!

Example: The 16-bit instruction x1380 specifies the operation M[x3] ← x8 (i.e. the microprocessor must write the value x8 into the RAM address x3).

ARCHITECTURAL OVERVIEW

In order to standardize the implementation of the device, the architectural details specified below must be followed. This DDmini must be implemented using a design decomposed into data unit and control unit using the control and status signals show below. The microprogrammed control unit will then be implemented to sequence the datapath through the necessary microperations to execute each instruction given to the microprocessor.

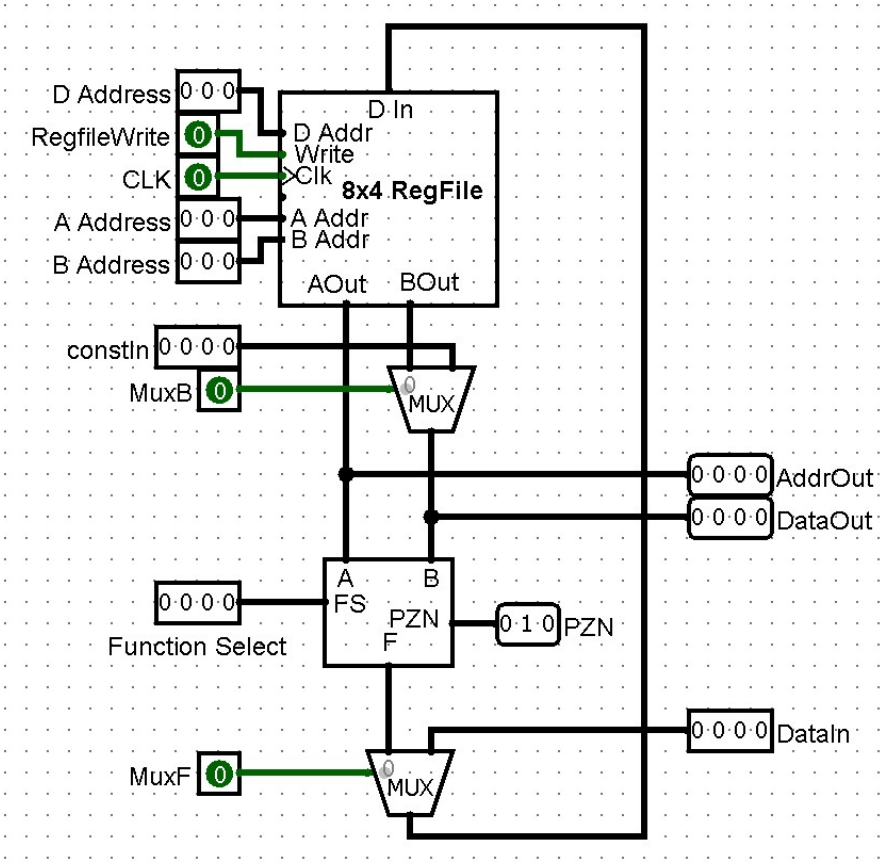


LAB 11 – DATAPATH CONSTRUCTION

The goal of the first week is to implement the datapath for the DDmini. The datapath consists of two primary components (a register file and a function unit) and the glue logic between them.

- [2 points] The first primary component is an 8 x 4 register file with a dual-ported read and a single write port. In other words, this device contains eight 4-bit general purpose registers for which two read and one write operation can be performed simultaneously. Implement the register file, label the inputs appropriately, and prepare it for use as a subcircuit in the datapath.
- [2 points] The second primary component of the datapath is a function unit capable of performing the operations listed in the table below.
    a) In addition to the output F from the function unit, the Function unit must also output a status signal PZN that indicates if the currently output of the function unit is positive (PZN = 100), zero (PZN = 010), or negative (PZN = 001).

FS	Microoperation
x0	F = 0
x1	F = 1
x2	F = A
x3	F = B
x4	F = A + B
x5	F = A – B
x6	F = A * B
x7	F = A / B
x8	F = NOT(A)
x9	F = A AND B
xA	F = A OR B
xB	F = NOT(A) + 1
xC	F = A >> 1
xD	F = A << 1
xE	F = A >>> 1
xF	Reserved



- [2 points] Implement the datapath using your register file, your function unit, and appropriate multiplexer logic. Modify and prepare your datapath to be used as a subcircuit in the DDmini implementation:
    a) Organize the control signals for your datapath into one 16-bit control word. The fields for the datapath control word are defined as:

15:13	12	11:9	8	7:4	3:1	0
AA (3)	MB (1)	BA (3)	MF (1)	FS (4)	DA (3)	RW (1)

b) In addition to the control word, the data path has two 4-bit inputs: Constant In (constIn[3:0]) and Data In (DataIn[3:0]). A single-bit input CLK should be used as the clock signal for the register file. The datapath has three outputs: Address Out (AddrOut[3:0]), Data Out (DataOut[3:0]), and the status signals PZN.

- Test your datapath throughout and document your implementation, design decisions, and test strategies in your labbook.
  a) Assume that all words in RAM memory are initialized to x0. In your lab notebook, show the expected results (i.e. the contents of RAM memory) of the execution of the following sequence of microprocessor instructions: x1030, x1120, x3001. Remember these words are instructions to the microprocessor not control words to the datapath.
  b) Note that none of the microprocessor instructions directly access the registers in the datapath. The ISA for this machine consists of *memory-to-memory operations* only. We will, however, need to use the registers in our datapath to perform these instructions through a sequence of microoperations. Since the "external system" cannot directly access our registers we are free to use all of them as scratch (temporary) registers during the micro-routine for each instruction. For each of the microprocessor instructions given above, determine a sequence of datapath microoperations that implement the specified microprocessor instruction. Verify that your datapath can execute these operations by applying the appropriate control word.

LAB 11 - DEMONSTRATION

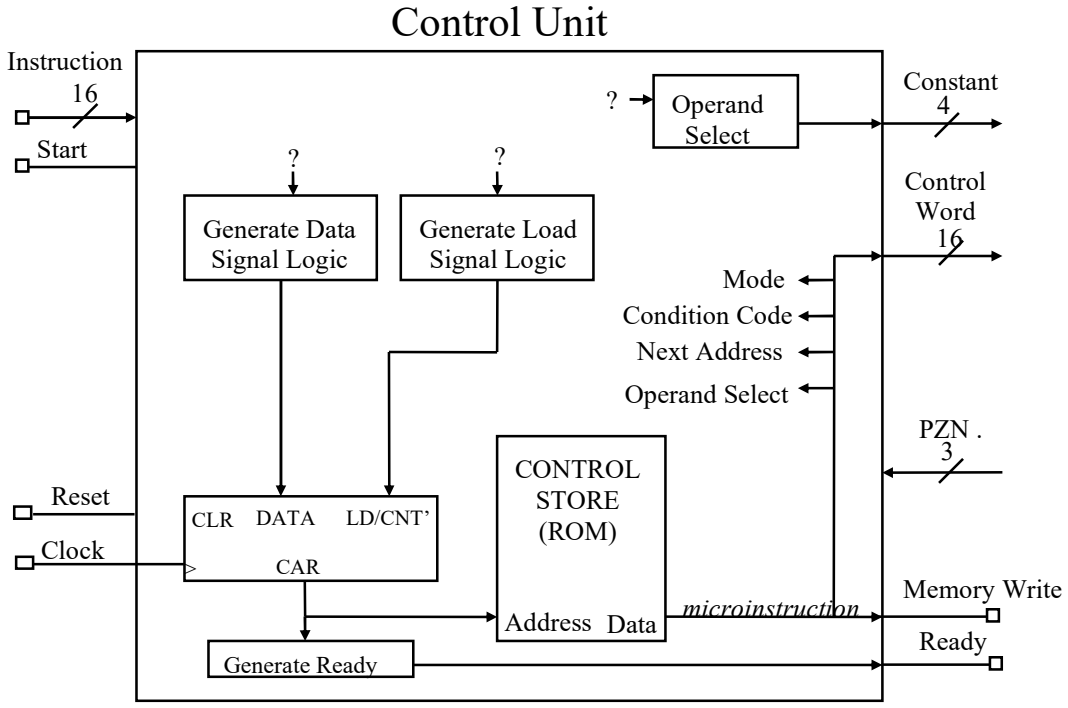
[2 points] Demonstrate your datapath implementation to your lab instructor. Be prepared to answer questions about the datapath such as "What microoperation is implemented for a given datapath control word", and "What change should we expect to see in the registers for this microoperation". Be prepared to determine (by hand) a sequence of datapath operations to execute ANY given microprocessor instruction. Be prepared to demonstrate the correct execution of your sequence of operations on your simulation. Be prepared to answer questions regarding your documentation and the design process.

Integrated Writing [2 points]: Refer to "Digital System Design: Engineering Journals & Lab Policies" for details. 0.5 points each for Completeness, Clarity, Organization, and Testing.

LAB 12 – CONTROL UNIT CONSTRUCTION

This week's goal is to complete the implementation of the multi-cycle computer by implementing a control unit and memory subsystem. The control unit for this computer will be a *non-programmable microprogrammed control with implicit sequencing*. In other words, the control unit does not have a program counter; the instructions that it needs to execute next are provided *to* it by the external system (in this case – the student).

Each of the instructions to the computer will require the execution of a sequence of microoperations. The microoperations performed for each instruction will be determined based upon the contents a microprogram in the control store. Unless the contents of the control word explicitly contain a *branch,* the next instruction will be the next address in the control store (e.g. implicit addressing). In this lab, students will design a control address register (CAR), a control store, and decoding logic for the control unit and then design the microprogram (a process known as microcoding).



The control unit will use the inputs for INSTRUCTION, START, RESET, CLOCK (CLK), and the datapath status bits P, Z, and N (PZN). The control unit will have outputs for READY, MEMORY_WRITE (MW), CONSTANT (CONST), and the datapath CONTROL_WORD (CW).

The primary components of the control unit are a control store and the control address register (CAR). The control store is a ROM containing the microinstructions for the control unit. These microinstructions determine the sequence of microoperations performed by the data unit. The CAR contains an address that is used as the input to the control store to select the "current" microinstruction. The logic that generates the data and load signals for the CAR controls the sequencing of the control unit microinstructions.

The architects for this system have defined the IDLE state to be the state in which CAR = x00. When CAR contains the value x00, READY should be asserted and the control unit should begin the processing of a new instruction when START is asserted. The microroutine for each instruction will begin at address INSTRUCTION[15:12]0000 (i.e. The microprogram for the instruction with opcode E should begin at

control store address xE0, where the opcode value is taken from the 4 most significant bits of the instruction).

The architects for this system have determined that this computer will use a 256-address (8-bit address) control store and have defined two classes of microinstruction. Each class of microinstruction will decode the microinstruction fields in a different way. This is advantageous as it decreases the number of bits required for each microinstruction (and therefore decreases the necessary size of the ROM). The MSB of the microinstruction will be used as a MODE bit to indicate the type of instruction and aid in the decoding.

The first class of microinstruction will be used to control the datapath by determining the control signals outputs of the unit. In addition, one bit in this class of microinstruction (MW: Memory Write) will be used to control the Write Enable output of the control unit.

The second class of microinstruction will be microsequencing/branch instructions used to perform the explicit sequencing of the control unit. The sequencing microinstruction consists of two fields. The first is a 3-bit field CONDITION and the second is an 8-bit field NEXT_ADDRESS (note that this is sufficient to select any address in the control store). If the condition specified by CONDITION is true then the microinstruction at NEXT_ADDRESS must be executed next (CAR ← NEXT_ADDRESS) otherwise the next microinstruction is implicitly chosen (CAR ← CAR + 1).

CONDITION	BRANCH IF
x0	0 (NEVER)
x1	N
x2	Z
x3	N OR Z

CONDITION	BRANCH IF
x4	P
x5	P OR N
x6	P OR Z
x7	ALWAYS

Note that the branches made by microsequencing instructions are determined by the status bits set by last value to pass through the function unit during a datapath microinstruction. These values will need to be saved (in the control unit) for potential use in any microsequencing instructions that might follow.

The first class of microinstruction requires 18 bits (1-bit mode + MW + 16-bit control word). The second type of instruction requires 12 bits (1-bit mode + 3-bit condition + 8-bit address). The minimum size for each stored microinstruction is therefore 18 bits. The architects for this system have attempted to define the fields for the branch microinstructions to simplify decoding. Bits 16:15 and 3:0 of every microsequencing instruction must always be set to 0. These bits include the write controls (MW and RW) to our memory subsystem and data unit. Thus, we can treat all of the other control signals of the control unit as *don't cares* during a branch instruction; regardless of the controlword to the datapath, no state change occurs in the datapath or RAM if RW and MW are 0 (respectively). The microinstruction formats follow:

Fields for microinstructions that control a microoperation (MODE = 0):

17	16	15:0
Mode=0	MW	Control Word (16)

Fields for microsequencing instructions (MODE=1):

17	16:15	14:12	11:4	3:0
Mode=1	00	CONDITION (3)	NEXT_ADDRESS (8)	0000

LAB 12 - IMPLEMENTATION

- Create a new circuit (CAR). The control address register for this device needs to be an 8-bit register (since the control store has an 8-bit address) with the ability to increment (to allow an implicit next instruction), load (to allow an explicit next instruction) and clear (to create an initial state). Design such a circuit with active high synchronous inputs CLR and LOAD, data input CAL[7:0] (Control Address Load), clock input CLK, and data output CA[7:0].
- Design and implement the logic necessary to generate the load and data signals for the CAR. The CAR will sometimes need to be loaded with the value of NEXT_ADDRESS, sometimes it will need to be loaded with an address based upon the OPCODE of the current instruction, and sometimes the CAR will need to be incremented. This correct behavior is based upon the current MODE bit, CONDITION FIELD, OPCODE, NEXT_ADDRESS, and the value of START.
- The CONSTANT_OUT signal must be able to deliver the values of OPERAND A, OPERAND B, or OPERAND C depending upon the needs of the current microinstruction. The architects have noted that the value of BA is a *don't care* when the datapath is loading a constant. Therefore, we will use the BA field to select *which* of the instruction operands is sent to the datapath when loading a constant. Design and implement the logic necessary to generate CONSTANT_OUT using the following encoding:

BA	000	001	010	011
CONSTANT_OUT	x0	OP A	OP B	OP C

- [2 points] Using the sequencing logic for the CAR, the logic for generating the CONSTANT_OUT SIGNAL, the CAR, the Control Store, and other necessary logic, complete the design of the microprocessor control unit.

- Create a new circuit schematic for your overall microprocessor. Create inputs and an output probe representing the system inputs and outputs described (above) for the overall computer. Create instances of your control unit, data unit and memory subsystem and connect them appropriately. Include the design in your lab book. Be sure to include schematic printouts and simulation/test results for all of your circuits.
- Create a new circuit schematic for system memory. The datapath has a 4-bit Address Out signal and 4-bit Data In/Out signals. The computer can therefore utilize a 16 x 4 RAM as its data memory subsystem. Implement the memory system for your computer. In addition to the ADDRESS and DATA IN inputs the memory store requires an input for WRITE enable. The only output is the 4-bit DATA OUT signal. Create an appropriate memory for your system.

- [1 point] Create a new main top-level circuit schematic that includes your overall microprocessor and the system memory. Create inputs and an output probe representing the system inputs and outputs described (above) for the overall computer.

LAB 12 – MICROPROGRAM DESIGN

The final goal of Digital Design laboratory is to complete the implementation of the control unit by microprogramming the control store.

Consider the operation M[A] ← B. This operation can be implemented using a sequence of RTL microperations available to our data unit. An example of one possible implementation of this instruction would be the sequence of transfers:

R0 ← CONSTANT (OP A)
R1 ← CONSTANT (OP B)
Address Out ← R0, Data Out ← R1 (MW)
CAR ← x00

- [3 points] Program your control store
  - Construct a table in your labbook that corresponds to the states/addresses and micro operations necessary to implement all of the instructions specified for this microprocessor.
  - Recall that the primary state information in the control unit is the contents of the CAR. This computer uses the value of OPCODE to determine the next start when START is asserted. Thus, the beginning of the microprogram will be found in the control store at address xN0 where N is the 4-bit opcode for that instruction. Each instruction will require one or more microinstructions, each of which requires its own state and thus a subsequent entry in the control store. Don't forget to include a sequencing command to return the CAR to the idle state once the instruction has been fully executed.
  - Include in your labbook a complete detail of each ROM address (in HEX) followed by the contents at that address (in RTL, in binary, and in HEX).
  - Update your computer's control store ROM to the values that you have determined above. Test each computer instruction thoroughly and correct any errors.

LAB 12 – DEMONSTRATION

Testing and Debugging consumes the majority the man-hours involved in most modern design projects. Expect to spend significant time testing and debugging each circuit and microprogram that you create. Include those tests in your labbook. *Be careful to test each component of your design thoroughly before including it into the next level of the hierarchy.*

If your circuit produces unexpected results from a previously tested subcircuit, first make certain that you have correctly connected the device. Drilling down into the subciruit may help to identify the problem.

[2 points] Demonstrate your computer and your proof of correctness (testing) to your lab instructor. Be prepared to answer questions about the control unit and to demonstrate its performance. You should be ready to point out how your circuit implements the functionality of the control unit, how you designed your microprograms, and how your microprograms implement the functionality of each instruction. You should be ready to simulate a sequence of instructions on your computer. You should also be ready to implement (via microcode) any new instructions (with Opcodes x9-xF) specified by the lab instructor and demonstrate its functionality during the final lab or course final.

Integrated Writing [2 points]: Refer to "Digital System Design: Engineering Journals & Lab Policies" for details. 0.5 points each for Completeness, Clarity, Organization, and Testing.

If you have made it to here, you have survived DIGITAL SYSTEMS DESIGN. Congratulations!