ELEC 5200 CPU Project 2

Noah Niedzwiecki

February 16, 2019

- The following will describe the datapath design for the of the CPU that accommodates the Niedzwiecki ISA. This processor is 16-bit and can execute 16 unique instructions which are specified by the 4 most significant bits in an instruction bit-sequence. These four bits feed into an instruction control circuit which enables almost every other circuit in the processor. Figure 1 below shows a block diagram detailing how each of the CPU's components connect.

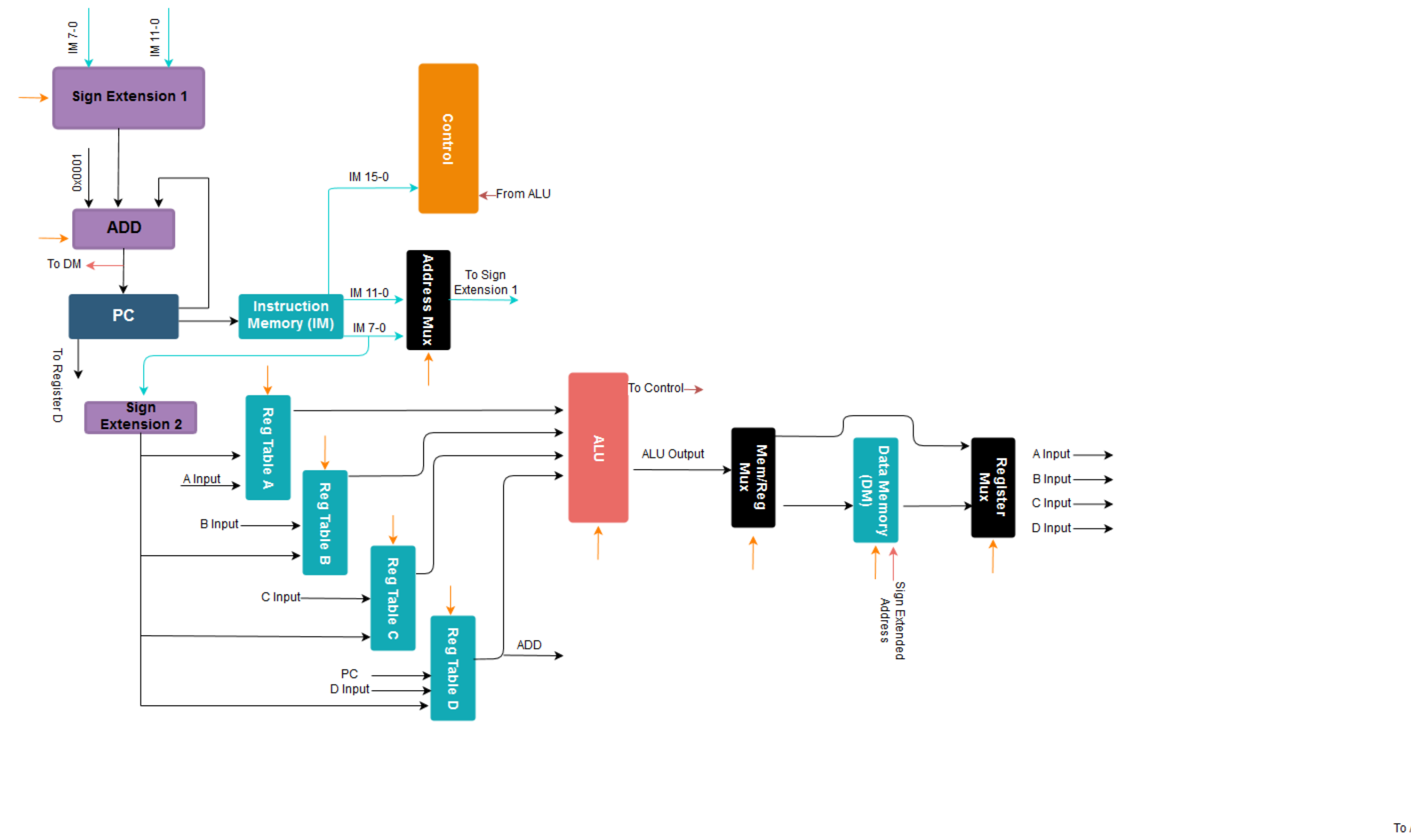

Figure 1: Niedzwiecki CPU Datapath Diagram

	15	14	13	12	RegisterMux	DataMemory	Mem/Reg Mux	ALU	Addr Mux	ADD	SignExtension1	RegTables
HALT	0	0	0	0	0	0	0	0	0	1	0	0
JUMP	0	0	0	1	0	0	0	0	1	1	1	0
ADD	0	0	1	0	1	0	1	1	0	0	0	1
SUB	0	0	1	1	1	0	1	1	0	0	0	1
LOAD	0	1	0	0	1	1	1	0	0	1	1	1
LOADC	0	1	0	1	0	0	0	0	0	0	0	1
MOV	0	1	1	0	1	0	1	1	0	0	0	1
STORE	0	1	1	1	0	1	0	1	1	1	1	1
CALL	1	0	0	0	0	0	0	0	1	1	1	1
RETURN	1	0	0	1	0	0	0	0	0	1	0	1
CALL if EQUAL	1	0	1	0	0	0	0	1	0	1	0	1
CALL if !EQUAL	1	0	1	1	0	0	0	1	0	1	0	1
OR	1	1	0	0	1	0	1	1	0	0	0	1
AND	1	1	0	1	1	0	1	1	0	0	0	1
CALL LESS THAN	1	1	1	0	0	0	0	1	0	1	0	1
CALL GREATER THAN	1	1	1	1	0	0	0	1	0	1	0	1

The table above shows each instruction's opcode and necessary activated circuit blocks.

The following section will go into detail for each instruction.

- HALT: With opcode 0000b, this instruction cancels cpu operation. To do this, the adding circuit block outputs address 0x0000 to the program counter which effectively stops all operation.

- JUMP: With opcode 0001b, this instruction adds a 12 bit offset to the program counter. To do this, the 4 MSB of the instruction set the control unit so that the Address Mux selects the 12 bits from the instruction and sends them to sign extension 1. The 16 bits resulting value sent to the adder circuit which adds the program counter and the offset together and stores it into the program counter.

- ADD: With opcode 0010b, this instruction adds 2 registers together and stores the sum in another register. The control unit realizes the opcode and determines which registers need to be accessed. Each register table holds 4 registers so 2 bits of the control line going into these tables act as the address. The Arithmetic Logic Unit has internal buffers which allow for serial data input from the registers in case the 2 referenced registers are in the same table. The control unit enables the ALU and selects which of the 4 arithmetic commands to execute on the input. The output is connected to the mem/reg mux which would then send it to the register mux which would then write to the proper register. Once the ALU operation concludes the control unit would then enable writing on the destination register and provide the necessary address.

- SUB: with opcode 0011b, this instruction executes the exact same way as add in terms of control unit flow. However, this operation sends single bit to the control unit to signify it a subtraction instruction results in a zero or negative value.
- LOAD: with opcode 0100b, this instruction populates a specified register with the contents of a memory address at the program counter plus an 8 bit offset. To do this, the 8 offset bits are sent through the address mux to the sign extender and eventually added with the PC. Instead of being loaded into the PC, this value is sent to the data memory module which loads the address value onto the register mux to be sent to the appropriate register file location.
- LOADC: with opcode 0101b, this instruction loads a register with an 8 bit immediate value. The 8 bits are sign extended with the sign extension 2 module and the result is then loaded into the proper register. Again, the register write and address are specified by the control unit signal.
- MOVE: with opcode 0110b, this instructions copies the value of one register into another. For this to take place, the register being read is loaded into the ALU. Then sent to the register mux through the mem/register mux selector. The register being written to is enabled for such a function and the write is complete.
- STORE: with opcode 0111b, this instruction takes a register's contents and saves it in the data memory at a location 8 bits offset from the PC. First the register is put into the ALU and sent to he mem/reg MUX to the data memory. To determine the data memory location, the 8 bit offset is sent through the address mux to sign extension 1. The value is added with the PC and the result is connected to the data memory module as the address to write to.
- CALL: with opcode 1000b, this instruction moves the current PC value to register rP and loads a new PC value according to the 8 bit immediate PC offset. The 8 bits are sent through the address mux to sign extension 1 and then added to the PC. To save the program counter, register table D allows for the data on the program counter line to be written to the last register available. Reg Table D is the only register table that has a line connected to the program counter for it is the only register that should be used to save the PC.
- RETURN: with opcode 1001b, this instruction moves the address saved in register rP and pushes it to the program counter. To do this, register rP is output to the ADD unit where 0 is then added and shifted onto the program counter. The control unit shuts the ALU off so that the value is not

altered in any way. A control signal going into the ADD unit identifies this as the one time that the data should be unmodified and added with 0.

- CALL if EQUAL: with opcode 1010b, this instruction evaluates two register values and determines if they are equal by subtracting the two and sending the appropriate signal to the control unit. If the difference between the two register values is zero then the CALL sequence will be executed.
- CALL if NOT EQUAL: with opcode 1011b, this instruction evaluates two register values and determines if they are equal by subtracting the two and sending the appropriate signal to the control unit. If the difference between the two register values is zero then the CALL sequence will not be executed.
- OR: with opcode 1100b, this instruction performs a bitwise OR on two registers' values and saves the result the specified address. This operation uses all the same modules as ADD instruction, but the the control unit instructs the ALU to perform the bitwise operation.
- AND: with opcode 1101b, this instruction uses the same components as OR, but the ALU is instructed to perform the logical AND operation on the two register operands.
- CALL LESS THAN: with opcode 1110b, this instruction behaves the exact same as the CALL if EQUAL command. If the reference register is smaller than the second register then the result will be zero or negative when the two are subtracted. The control unit is informed by the ALU and the CALL takes place.
- CALL GREATER THAN: with opcode 1111b, this instructions is the exact same as CALL LESS THAN, but it only occurs if the ALU provides the control unit a non-activated signal line

---

Each of the modules in figure 1 in detail:

- Control: This unit provides all of the enables and addresses selection bits throughout the CPU. It takes every bit of the instruction memory address value and determines which modules should be activated. The control unit also looks at which registers are being referenced in a given instruction and provides the corresponding address and enable to the register tables.
- Sign Extension 1: This module takes either 8 bits or 12 bits of data and extends the sign to a full 16 bits so it represents an address. In order for the ADD unit to do its calculations, the two operands

must be the same dimension.

- ADD: This is a simple adding circuit that adds two addresses together so they can be sent to the program counter or the address register rP. The control block determines if the sign extended value should be added in case of a jump or if the PC should just increment by 1. Additionally, the adder output is connected to the data memory address bus.

- PC: The program counter is the unit that controls which instruction from the instruction memory should be executed in the CPU. Typically, the program counter increments by 1 every clock cycle, but in the case of a jump, the add circuitry or rP register can load non sequential addresses into it.

- Instruction Memory: This memory has 2^{16} addresses than can hold 16 bits of instruction data per location. The program counter points to a memory location and its values are sent to the control unit, address mux, or register files.

- Address Mux: This allows for 2 different length operands to be loaded into the sign extension 1 unit. A single control bit determines which let through to be sign extended since there are only 2 options.

- Sign Extension 2: This module takes 8 bits from the instruction memory and extends them to 16 bits to be loaded into the register files. This doesn't need a control bit since the data wont be loaded into the register files unless the control unit explicitly enables the write line on the register file.

- Reg Table A-D: Each table contains 4, 16 bit registers. The control line here provides a read/write enable line, 2 read address lines, and 2 write address lines. The read lines determine which of the inputs it should accept when the write is enabled and the write address lines provide the address in which the write should occur in the file.

- ALU: This unit provides all the register arithmetic in the CPU. It can execute add, subtract, bitwise or, and bitwise and operations. It provides a bit of data to the control unit that acts as a zero flag.

- Mux/Reg Mux: This multiplexer takes the output from the ALU and determines if it should be stored in memory or sent back to the registers. One bit from the control unit determines which is selected.

- Data Memory: This memory holds all the saved values that the program generates. It comprises of 2^{16} addresses that are 16 bits in dimension. This can be read and written to based on the 1 bit control signal it is supplied.

- Register Mux: This multiplexer can either take data from the data memory or from the mem/reg mux. This unit is used for when data should be sent back to the register files. Each output is linked to a register table. Two control signal lines determine which of the 4 outputs the data is uploaded onto.

Comments:

- Harvard architecture was selected due to its simplicity.
- A single cycle design was chosen also due to its simplicity
- In this CPU, the control unit is going to be quite complicated. I tried to make everything feed into each other as immediately as possible so visualization and mapping would be simple.
- The decision to separate the registers into 4 different files was an effort to speed things along in the case that the referenced registers were in different files. If the programming is done correctly, the values could be shifted onto the ALU in a parallel fashion. This would probably result in higher power consumption and a larger footprint but that design aspect is not critical in this case.
- All of the registers can be latching due to the complexity of the control signal logic.
- The AND and OR bitwise operations were edited to include a destination register instead of rewriting one of the two supplied. This is just an effort to simplify programming later.