UF Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

Dr. Christophe Bobda
Newman Waters
EEL3701C - Fall 2025

HW7 Solutions

Part 1:

a. Given a 16-bit instruction 0011110101010000, identify the instruction that is being performed. Explain what the instruction is doing (what registers are being accessed/modified/stored to, what style of instruction is this, etc.).

Opcode: 001, ADD
Rs = 111 = R7
Rt = 010 = R2
Rd = 101 = R5
Remaining bits are unused
R5 = R2 + R7, register-style instruction

b. Why is the sign extension unit in *Figure 1* necessary for immediate instructions? Why would there be an issue if we didn't sign-extend the immediate/address?

The immediate field is only 7 bits large, so trying to perform math between a 7-bit number and a 16-bit value from a register will cause issues. The sizes of the buses should match, and the sign extension unit just pads the 7-bit value to become 16-bits but keep the same value.

c. For a *Load* instruction, what does the CPU need to do, step-by-step, to fetch a value from memory into a register based on *Figure 1*? What about the *Store* instruction?

Load: take instruction from instruction register, decode signals, pass the address into the memory module that we want to read (execute), then write the value from memory back into a register.

Store: take instruction from instruction register, decode signals, immediately send value to write to the data memory while calculating the write address, then store the value in data memory (no write-back).

UF Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

Dr. Christophe Bobda
Newman Waters
EEL3701C - Fall 2025

Part 2:

a. Based on the way the program memory will be laid out (1 16-bit instruction at each address), how much do you need to increment the program counter each clock cycle?

Increment by 1 per clock cycle

b. Thus, how many clock cycles does it take to execute a single instruction with this methodology? Why is this method not necessarily efficient, and how could we potentially improve execution time?

1 clock cycle. Use pipelining for better resource allocation. This is inefficient because the clock cycle needs to be at least as long as the time it takes for the longest instruction (load word) to execute.

Part 3:

a. Inspect the following diagram, and based on the description above, combine the ***Branch*** control signal and the **Z** flag with some logic to choose between the processor going to the next instruction (Mux select 0), or the address to branch to (Mux select 1) (Draw out signals and logic gates/components).

AND gate with Branch and Z signal going into the multiplexer is sufficient

b. The instruction set does not specify a comparison instruction. With our *very* limited instruction set, how would you perform a comparison operation to prepare for a branch instruction (conditional code)?

For example, if checking that a value in register is 5, you can do 5 + (-5) = 0 (subtraction), and if the zero flag is asserted as a result of this signed addition operation, we know that the two values are equal.

c. Based on the design of the instruction, how many addresses could we possibly jump to (i.e. how big should our Program/Instruction memory be at maximum)?

2^7 address because the address field in the immediate style instructions is 7 bits.

d. Explain the difference between conditional and unconditional jumps.

Conditional jumps occur only when a certain condition is met, while unconditional jumps will occur no matter what as soon as the instruction is read.

e. Can you think of any potential dangers of using the jump function (think of processor security)? What do you think would happen if a processor were to try to jump to a location outside of the normal address space on a real computer?

If an attacker somehow managed to change the jump address in the code, they could make your seemingly innocent program jump to a piece of malicious code without you knowing. Generally, there is address space protection on computers to prevent things like this from happening (can get a segmentation fault if accessing memory outside of what is normally allocated to the program by the OS).