# Design and Programming of the M270 Computer

**You will learn how to design the control unit of a simple computer and how to program it in "machine" code.**

## 1.0 Introduction

While EECS 270 is not meant to turn you into a computer designer, it does actually provide enough background in digital design to at least let you tackle the design of a simple stored-program computer. This final lab experiment is concerned with understanding the basic organization of a computer. You are given an instruction set architecture (ISA) and a corresponding datapath and are asked to design the controller that will implement the ISA on that datapath. To appreciate what you've accomplished, you are also asked to write a simple program that can be executed on "your" computer.

## 2.0 Preparation

- Chapter 9 of Hayes (one of our reference texts) gives a good overview of computer organization and basic architecture. Read it.
- The M270 instruction set and its particular implementation in this project will be discussed at length in class. Do not skip these lectures!

## 3.0 Design Specifications

M270 is an 8-bit computer with the following 16-bit instruction format:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OpCode | | | | Ra | | Rb | |
| n | | | | | | | |

where:

- OpCode is a 4-bit field that encodes 16 different instructions
- Ra is a 2-bit destination register field
- Rb is a 2-bit source register field, and
- n is an 8-bit two's complement immediate operand

M270's instruction set architecture (ISA) is based on an 8-bit address allowing programs to access 256 bytes of primary memory (i.e. addresses 00 to FF in hexadecimal). Source operands are designated by Y and determined using three addressing modes:

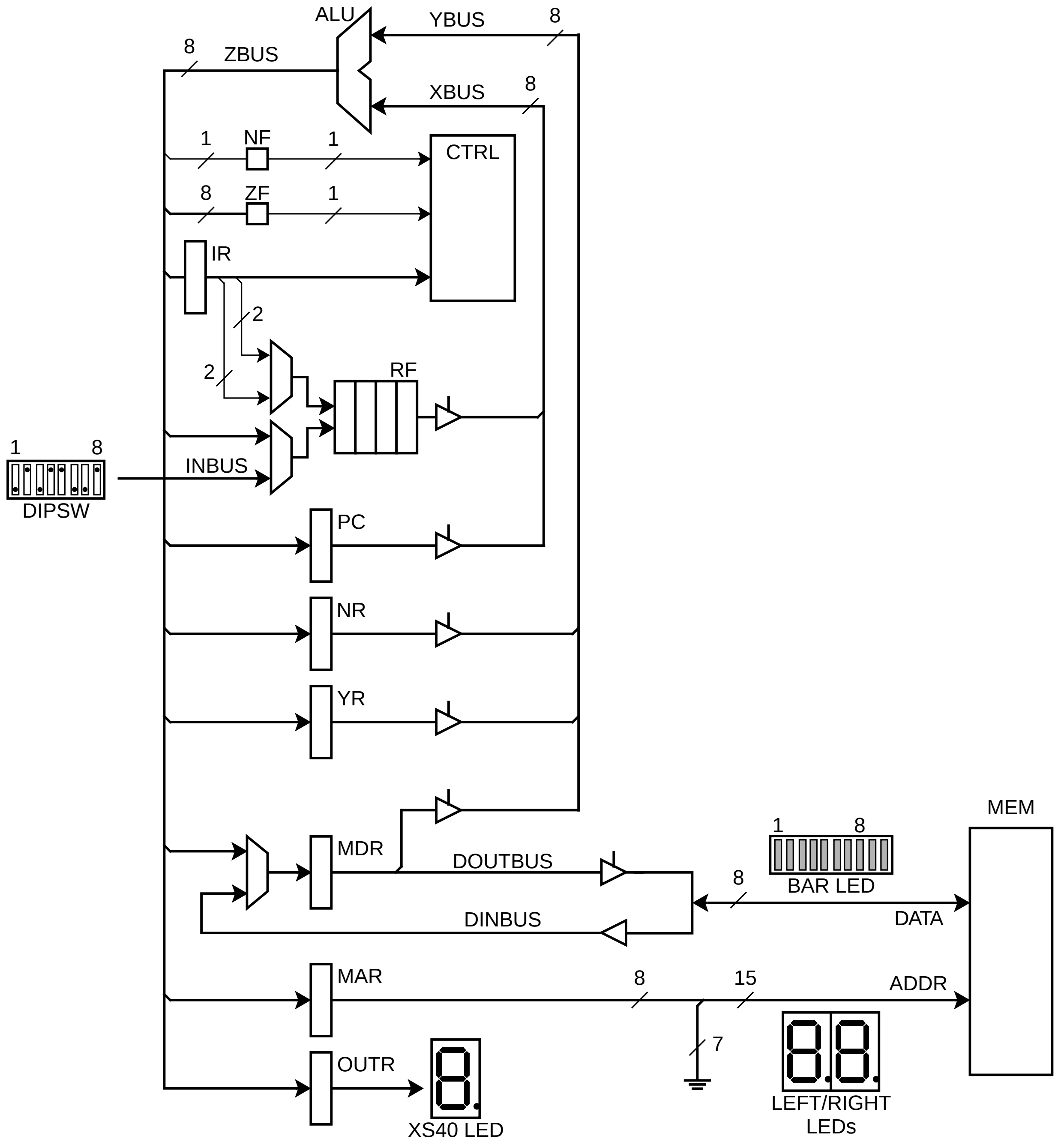
- **Immediate**: Yi = n
- **Register**: Yr = Rb + n (i.e. the contents of Rb plus n)
- **Memory**:Ym = MEM[Rb + n] (where MEM denotes the primary memory array)

### 3.1 Datapath Architecture

M270 has the bus-oriented datapath architecture shown in Figure 10. The primary CPU busses are the operand busses **XBUS** and **YBUS** and the results bus **ZBUS**. The interface to primary memory consists of an address bus **ADDR** and a bidirectional **DATA** bus; internally, the DATA bus is split into a **DINBUS** and a **DOUTBUS**. The major components interconnected by these busses are:

- **ALU**: a multi-function arithmetic and logic unit.
- **NF** and **ZF**: condition flags that get set to indicate that the result of an arithmetic operation in the ALU is, respectively, negative and zero.
- **IR**: the instruction register which holds the first byte (OpCode, Ra, and Rb) of the current instruction.
- **RF**: a register file consisting of 4 8-bit registers: R0, R1, R2, and R3.
- **PC**: the program counter
- **NR**: a register that holds the second byte of the current instruction (the immediate operand n).
- **YR**: the source operand register that holds either Yr (register operand) or Ym (memory operand).

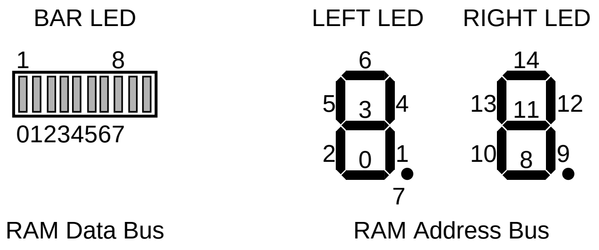
**FIGURE 10.** The M270 Datapath



- **MDR**: the memory data register which acts as the data interface to primary memory.
- **MAR**: the memory address register which acts as the address interface to primary memory.

The I/O interface to M270 is through the dip switches DIPSW on the input side, and the XS40 LEDs on the output side: the settings of DIPSW can be loaded into any register in RF, and the contents of a special output register **OUTR** can be displayed on the XS40 LEDs (all except the most significant bit since the LED has just seven segments.)

Primary memory is a 32KB static RAM chip that resides on the XS40 board. The data and address lines of this RAM are connected to the BAR and LEFT/RIGHT LEDs on the XStend board as shown below:



These connections make the memory busses automatically observable and are useful for debugging. Note that the M270 ISA allows access to just the first 256 bytes of memory; hence MAR is connected to the least significant byte of the memory address bus (observable on the LEFT LED) whereas the seven most significant address bits are grounded.

## 3.2 Instruction Set

M270's instruction set is shown in Table 1. For each instruction, the table gives its mnemonic name, the values in each of its fields in hexadecimal, the effect of its execution using "register transfer level" (RTL) notation, and a brief explanatory comment. The instructions can be grouped into three distinct classes:

- **Program control** instructions that allow a program's normally-sequential control flow to be altered. These include three branch instructions (BRU, BRN, and BRZ) and the HALT instruction that terminates program execution.
- **Data movement** instructions for loading, storing, and displaying the contents of the register file:

| | |
|---|---|
| INP: | loads register Ra from the dip switches DIPSW |
| LDI: | loads register Ra with the immediate operand Yi = n |
| LDR: | loads register Ra with the register operand Yr = Rb + n |
| LDM: | loads register Ra with the memory operand Ym = MEM[Rb + n] |
| STR: | stores register Ra in memory location whose address is Rb + n |
| OUT: | displays register Ra on the XS40 LEDs |

- **Data processing** instructions that perform arithmetic (ADDR and ADDM) and logical (ANDR, ANDM, INVR, and INVM) operations on data.

**TABLE 1.** M270 Instruction Set (field contents shown in hex)

| **Mnemonic** | **First Byte** 7 6 5 4 | 3 2 | 1 0 | **Second Byte** 7 6 5 4 3 2 1 0 | **RTL Operation** | **Comment** |
|---|---|---|---|---|---|---|
| HALT | 0 | 0 | 0 | 00 | ____ | Stop execution |
| BRU | 1 | 0 | Rb | n | PC ← Yr | Branch unconditionally |
| BRN | 2 | 0 | Rb | n | PC ← Yr when NF = 1 | Branch when negative |
| BRZ | 3 | 0 | Rb | n | PC ← Yr when ZF = 1 | Branch when zero |
| STR | 4 | Ra | Rb | n | MEM[Yr] ← Ra | Store to memory |
| INP | 5 | Ra | 0 | 00 | Ra ← DIPSW | Input from DIPSW |
| OUT | 6 | Ra | 0 | 00 | S ← Ra | Output to XS40 LEDs |
| LDI | 7 | Ra | 0 | n | Ra ← Yi | Load immediate operand |
| ADDR | 8 | Ra | Rb | n | Ra ← Ra + Yr | Add from register |
| ANDR | 9 | Ra | Rb | n | Ra ← Ra & Yr | And with register |
| INVR | A | Ra | Rb | n | Ra ← !Yr | Invert register |
| LDR | B | Ra | Rb | n | Ra ← Yr | Load from register |
| ADDM | C | Ra | Rb | n | Ra ← Ra + Ym | Add from memory |
| ANDM | D | Ra | Rb | n | Ra ← Ra & Ym | And with memory |
| INVM | E | Ra | Rb | n | Ra ← !Ym | Invert memory |
| LDM | F | Ra | Rb | n | Ra ← Ym | Load from memory |

## 3.3 Register Transfer Operations

Instruction processing in M270 is accomplished by the orderly execution of a sequence of RTL operations. The RTL operations permitted by the M270 datapath architecture can be classified into the following three groups:

### 3.3.1 CPU Operations

These operations have the template

DST ← SRCX OP SRCY

where DST, SRCX, SRCY, and OP refer to a destination register, a source register on the XBUS, a source register on the YBUS, and an ALU operation, respectively. The possible "values" for each of these "variables" are:

| | |
|---|---|
| DST: | IR, RF, PC, NR, YR, MDR, MAR, OUTR |
| SRCX: | RF, PC |
| SRCY: | NR, YR, MDR |
| OP: | ADD, AND, INV, PASSX, PASSY |

Note that in addition to ADD, AND, and INV, the ALU allows the values on the XBUS and YBUS to be passed through to the ZBUS using the special ALU operations PASSX and PASSY.

### 3.3.2 Memory Operations

The CPU interfaces to primary memory through the memory data and memory address registers. The RTL operations corresponding to reading from and writing to memory are:

| | |
|---|---|
| READ: | MDR ← MEM[MAR] |
| WRITE: | MEM[MAR] ← MDR |

### 3.3.3 Input/Output Operations

These operations correspond to the INP and OUT instructions: for INP, an appropriate register in RF is the destination of a transfer from DIPSW; for OUT, the RTL operation OUTR ← RF[Ra] transfers the content of register Ra to the output register for display on the XS40 LEDs.

## 3.4 Enabling RTL Operations

The RTL operations described above can be enabled by setting appropriate control signals on the various datapath components. Table 2 lists the signals that can be used to control the operation of M270's datapath components. For example, the hypothetical RTL operation NR ← PC + MDR (which is not part of M270's instruction repertoire!) can be performed by making the following control signal settings:

| | |
|---|---|
| NR_LOAD = 1 | Enable the loading of NR from the ZBUS |
| PC_READ = 1 | Enable the reading of PC on the XBUS |
| MDR_READ = 1 | Enable the reading of MDR on the YBUS |

**TABLE 2.** M270 Control Signals

| **Component** | **Signal Name** | **RTL Operation** | **Comment** |
|---|---|---|---|
| ALU | ALU_PASSX | ZBUS ← XBUS | Pass value on XBUS to ZBUS |
| | ALU_PASSY | ZBUS ← YBUS | Pass value on YBUS to ZBUS |
| | ALU_ADD | ZBUS ← XBUS + YBUS | Add XBUS to YBUS |
| | ALU_AND | ZBUS ← XBUS & YBUS | And XBUS with YBUS |
| | ALU_CMP | ZBUS ← !XBUS | Invert XBUS |
| RF | RF_ASEL | RFADDR ← RF_ASEL? Rb: Ra | Select address of register in RF |
| | RF_DSEL | RFSRC ← RF_DSEL? INBUS: ZBUS | Select source of transfer into RF |
| | RF_READ | XBUS ← RF[RFADDR] | Read RF |
| | RF_LOAD | RF[RFADDR] ← RFSRC | Load RF |
| PC | PC_READ | XBUS ← PC | Read PC |
| | PC_LOAD | PC ← ZBUS | Load PC |
| | PC_INC | PC ← PC + 1 | Increment PC |
| | PC_CLEAR | PC ← 0 | Clear PC |
| IR | IR_LOAD | IR ← ZBUS | Load IR |
| NR | NR_READ | YBUS ← NR | Read NR |
| | NR_LOAD | NR ← ZBUS | Load NR |
| YR | YR_READ | YBUS ← YR | Read YR |
| | YR_LOAD | YR ← ZBUS | Load YR |
| MAR | MAR_LOAD | MAR ← ZBUS | Load MAR |
| MDR | MDR_SEL | MDRSRC ← MDR_SEL? ZBUS: DINBUS | Select source of transfer into MDR |
| | MDR_READ | YBUS ← MDR; DATA ← MDR | Read MDR |
| | MDR_LOAD | MDR ← MDRSRC | Load MDR |
| MEM | MEM_READ | DINBUS ← MEM[MAR] | Read from primary memory |
| | MEM_WRITE | MEM[MAR] ← MDR | Write to primary memory |
| OUTR | OUTR_LOAD | OUTR ← ZBUS | Load OUTR |

| | |
|---|---|
| ALU_ADD = 1 | Enable the ALU to perform an ADD operation |

It is important to note that the datapath architecture allows some RTL operations to occur simultaneously as long as they do not cause resource conflicts, such as control of a bus. Specifically, any ALU operation can be performed simultaneously with any memory or input operation. Additionally, the PC can be incremented independently of any other operation.

### 3.5 Instruction Cycle

The M270 ISA can be implemented by the following instruction processing cycle:

| Phase | RTL Operation(s) | Comment |
|---|---|---|
| Fetch | IR ← MEM[PC] | Load first byte of instruction into IR |
| | PC ← PC + 1 | Increment PC |
| | NR ← MEM[PC] | Load second byte of instruction into NR |
| | PC ← PC + 1 | Increment PC to point to next instruction |
| Generate Y | YR ← RF[Rb] + NR | Compute Yr, the “register” operand |
| | if memory reference instruction: YR ← MEM[YR] | Compute Ym, the “memory” operand |
| Decode / Execute | Perform RTL operation according to Table 1 | Decode and execute the instruction |

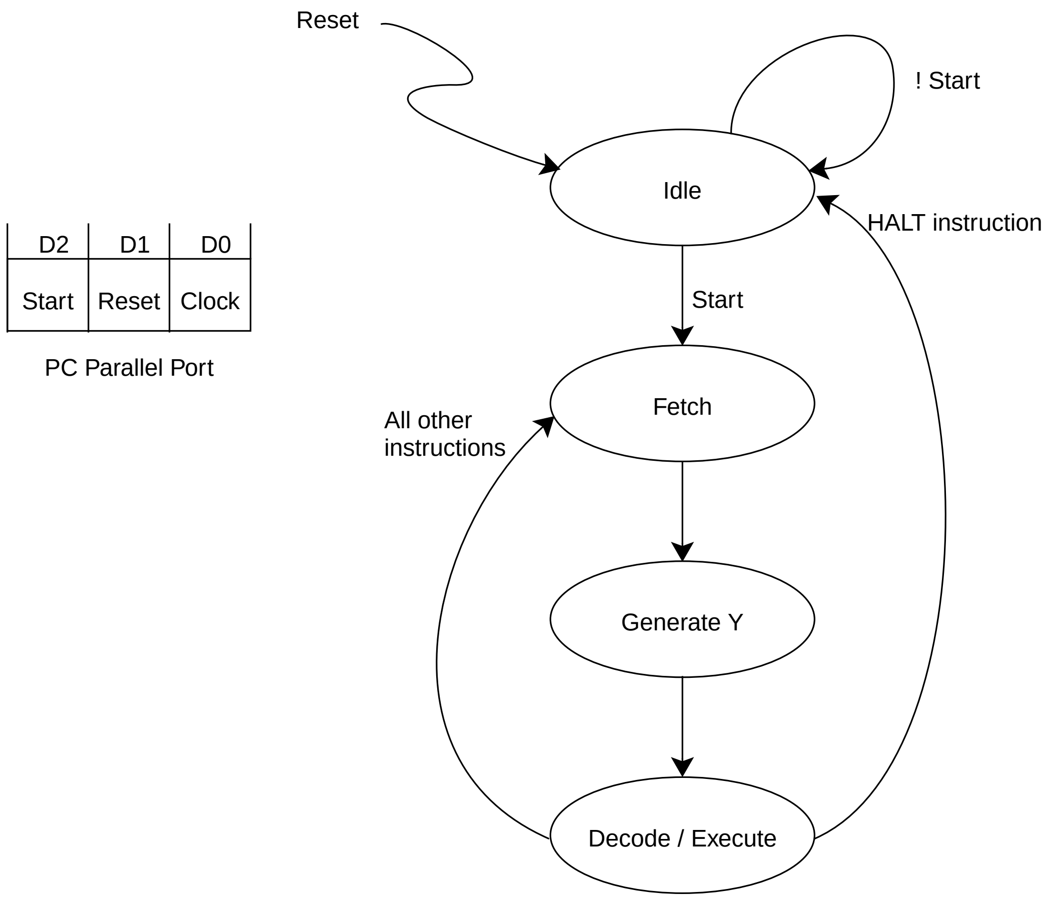
The M270 datapath architecture may not allow a direct single-cycle implementation of some of the RTL operations in the above table. For instance, memory can be accessed only through MAR and MDR. Thus the RTL operation IR ← MEM[PC] which loads the first byte of the current instruction into IR must be decomposed into the following three architecture-supported RTL operations:

| RTL Operation | Control Signal Settings |
|---|---|
| MAR ← PC | PC_READ = 1; ALU_PASSX = 1; MAR_LOAD = 1; |
| MDR ← MEM[MAR] | MEM_READ = 1; MDR_SEL = 0; MDR_LOAD = 1; |
| IR ← MDR | MDR_READ = 1; ALU_PASSY = 1; IR_LOAD = 1; |

The “top-level” state diagram of the M270 controller is shown in Figure 11. Its operation is controlled by three signals that are applied through the PC parallel port: Clock, Reset, and Start. Before normal operation starts, M270 must be “reset” by asserting the synchronous Reset input. This puts the computer into its Idle state with PC (among other registers) initialized to 0. Normal operation is indicated by asserting Start for one clock cycle; this initiates the fetch/generate Y/decode/execute instruction cycle. The computer returns to its idle state upon executing a HALT instruction.

Applying the clock signal from the PC parallel port is useful for single-stepping through program execution. If you feel that your implementation is correct and would like to run it “at-speed”, you may use the 12-MHz on-board clock by connecting M270’s clock input to pin 13.

**FIGURE 11.** Major phases of the M270 instruction processing cycle



## 4.0 Design Notes and Hints

- The RAM chip on the XS40 board can be “loaded” using the same GXSLOAD utility used to download the FGPA bitstream configuration file. Save the desired contents of memory in a text file with a .HEX extension and drag/drop the file on the GXSLOAD icon to load it into the RAM. The file should have one or more text lines formatted as follows:

  – <#bytes> <starting address> <byte 1> <byte 2> . . . <last byte>

  Note that the first character in each line is a dash, and that fields are hexadecimal numbers separated by spaces. For example, if we want to load the RAM with the four bytes 21 F3 5E 77 starting at address 110D (all values in hex), we would prepare the one-line .HEX file:

  – 4 110D 21 F3 5E 77

- An assembler for the M270 instruction set was written by R. Krishna. For details on how to use it, check with your lab GSI. This assembler simplifies the generation of "object" code (i.e. the .HEX file) even though for this simple ISA it is not too hard to generate object code manually.

## 5.0 Deliverables

A project containing the datapath of M270 has been created for your use. Check with your lab GSI on the logistics of copying this project into your own directory. The project consists of two top-level schematics: M270 and RAMPORT. The M270 schematic contains the entire datapath as well as an ABEL macro shell for the controller that defines its inputs and outputs. Your primary task is to write the ABEL code that implements this controller, and to debug it by running some test programs. The RAMPORT schematic contains the I/O interface to the RAM chip.

An additional schematic, called MEM, is also available. MEM uses "soft" RAM32X8 parts from the xc4000x library to simulate the actual RAM chip. While debugging your controller design, you will find it easier to use MEM instead of RAMPORT since you will be able to use the simulator to load and observe its contents. The MEM schematic should be removed and replaced with RAMPORT once you're ready to implement your design and download it to the logic board.

This is a three-week lab; the due date for each of the pre-lab deliverables is noted below.

### 5.1 Pre-Lab (30%)

1. Hardcopy of the ABEL file that implements M270's controller (10%; due: week of March 27.)
2. Hardcopy of a test program that checks each of M270's instructions. You may use the INP and OUT instructions to inject values into M270's registers and to observe the results of applying each of its instructions (10%; due: week of March 27.)
3. Hardcopy of a program that sorts the numbers 7, -15, 4, -2, 25 in ascending order. You may load these numbers into a location of your choice in memory by putting them directly in the HEX file that contains your program. Verify that your program is correct by displaying the numbers in sorted order using the OUT instruction (10%; due: week of April 3.)

### 5.2 In-Lab (30%)

By now, you should know what you're expected to do in-lab. For this experiment, you need to demonstrate that your controller correctly implements all of M270's instructions and that your sorting program works. For an extra 10% bonus credit, demonstrate the "at-speed" operation of your computer by using the 12MHz on-board clock.

### 5.3 Post-Lab (40%)

Prepare your lab report as described in the *EECS270 Laboratory Overview* handout. Make sure you complete and include all parts of the report including the *Cover Sheet*, the *Design Narrative* section, and the *Design Documentation* section. Include as part of your design documentation all corrected pre-lab requirements.

In your design narrative, explain any difficulties encountered and how you resolved them. Also, explain why the ZF and NF flip-flops are enabled by the ALU_ADD signal. Finally, describe how you would modify this datapath architecture to allow a two-cycle instruction fetch (one to load IR and one to load NR).