›

# Midterm Exam

```
 _____ _____ ____ ____    _____ _____ ___  
| ____| ____/ ___/ ___|  |___ /|___  / _ \ 
|  _| |  _|| |   \___ \    |_ \   / / | | |
| |___| |___| |___ ___) |  ___) | / /| |_| |
|_____|_____\____|____/  |____/ /_/  \___/ 
```

## EECS 370 Fall 2023: Introduction to Computer Organization

You are to abide by the University of Michigan College of Engineering Honor Code. Please sign below to signify that you have kept the honor code pledge:

***I have neither given nor received aid on this exam,***
***nor have I concealed any violations of the Honor Code.***

Signature: ______________________________

Name: ______________________________

Uniqname: ______________________________

Uniqname of person sitting to your ***Right***
**(**Write ⊥ if you are at the end of the row) ______________________________

Uniqname of person sitting to your ***Left***
**(**Write ⊥ if you are at the end of the row) ______________________________

**Exam Directions:**

- You have **120 minutes** to complete the exam. There are **9** questions in the exam on **18** pages (double-sided). **Please flip through your exam to ensure you have all pages.**
- You must show your work to be eligible for partial credit!
- Write legibly and dark enough for the scanners to read your answers.
- **Write your uniqname on the line provided at the top of each page.**

**Exam Materials:**

- You are allotted **one 8.5 x 11 double-sided** note sheet to bring into the exam room.
- You are allowed to use calculators that do not have an internet connection. All other electronic devices, such as cell phones or anything or calculators with an internet connection, are strictly forbidden.

| *Problem* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| *Point Value* | 10 | 15 | 10 | 8 | 8 | 10 | 12 | 7 | 20 |

## Problem 1: Multiple Choice 10 points

Completely shade in the boxes with the correct answers. Select only 1 answer. *[1 point each]*

1. While _______ has largely persisted to the current day, ________ is considered to have broken down in the mid-2000s, limiting how fast processors can be run.
   - □ Dennard scaling / Moore's law
   - □ Moore's law / Dennard scaling
   - □ Moore's law / Mealy's law
   - □ Mealy's law / Moore's law

2. Having variable length instruction encodings is considered a more common feature in ________-style architectures.
   - □ Single-cycle
   - □ Multi-cycle
   - □ CISC
   - □ RISC

3. Supporting an ADD instruction with 2 immediate values rather than an immediate and source register index, or two source register indices, is uncommon since:
   - □ There is no way to disambiguate if the value in an encoded instruction is a register index vs immediate
   - □ There would not be a way to handle negative offsets
   - □ Hardware cannot support sign-extending two numbers
   - □ The compiler / programmer can simply hardcode the sum in the program

4. Memory locations in the LEGv8 subset of ARM are exclusively ______ addressable
   - □ Byte
   - □ Word
   - □ Double-word
   - □ Big-endian
   - □ Little-endian

5. An ISA consists of 4 registers and 10 instructions. Every instruction is encoded into 16-bits, has fixed-sized opcodes, and either specifies 3 register indices or 2 register indices and an offset. Instructions can encode offsets up to _____ bits wide.
   - □ 2
   - □ 3
   - □ 7
   - □ 8
   - □ none of the above

6. Global variables are placed in the _____ section of memory
   - □ Stack
   - □ Heap
   - □ Static
   - □ Text

7. The clock period for a _______ datapath depends on how long it takes to execute the slowest instruction
   - □ Von Neumann
   - □ Single-cycle
   - □ Mealy
   - □ Multi-cycle

8. If the program counter of the LC2K processor ever points to a line in memory generated by the directive ".fill 29360128", the processor will: (hint: 29360128 = 7 << 22)
   - □ Perform an add instruction
   - □ Increment the PC without doing anything else
   - □ Crash
   - □ Write a value into memory

9. For which instruction latencies do we expect the multi-cycle datapath discussed in class to have the best improvement over the single-cycle datapath running the same workload?
   - □ All instructions have 1 ns latency
   - □ All instructions have 100 ns latency
   - □ 99% of instructions executed have 100 ns latency and 1% have 1 ns latency
   - □ 1% of instructions executed have 100 ns latency and 99% have 1 ns latency

10. A finite state machine with 16 states, 3 bits of input and 4 bits of output will require a control ROM with ______ bits to implement the next-state and output logic
    - □ 48
    - □ 128
    - □ 144
    - □ 512
    - □ 1024

## Problem 2: Short Answers 15 points

1. Convert each of the 8-bit hexadecimal numbers into its binary form and decimal form (both for treating the original number as a signed (two's-complement) and an unsigned value). *[5 points]*

| Hexadecimal | Binary | Decimal (signed) | Decimal (unsigned) |
|---|---|---|---|
| 0xA1 | 0b10100001 | -95 | 161 |
| 0x2C | 0b00101100 | 44 | 44 |

2. Consider an 8-bit floating point format based on the IEEE standard where the most significant is used for the sign, the next 3 bits are used for the exponent and the last 4 bits are used for the mantissa. The scheme uses "biased 3" to represent the exponent (rather than biased 127 used for a 32-bit IEEE floating point number) and has an implicit one just like the IEEE format. This scheme is called "VSF" (very short float).

| Sign | Exponent | Mantissa |
|---|---|---|
| 7 | 6 5 4 | 3 2 1 0 |

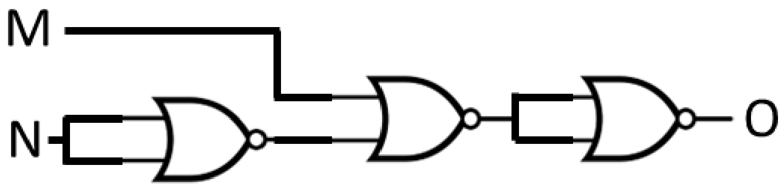
Write the *binary* encoding of 4.0 as a VSF number. *[3 points]*

0b___0 101 0000___________

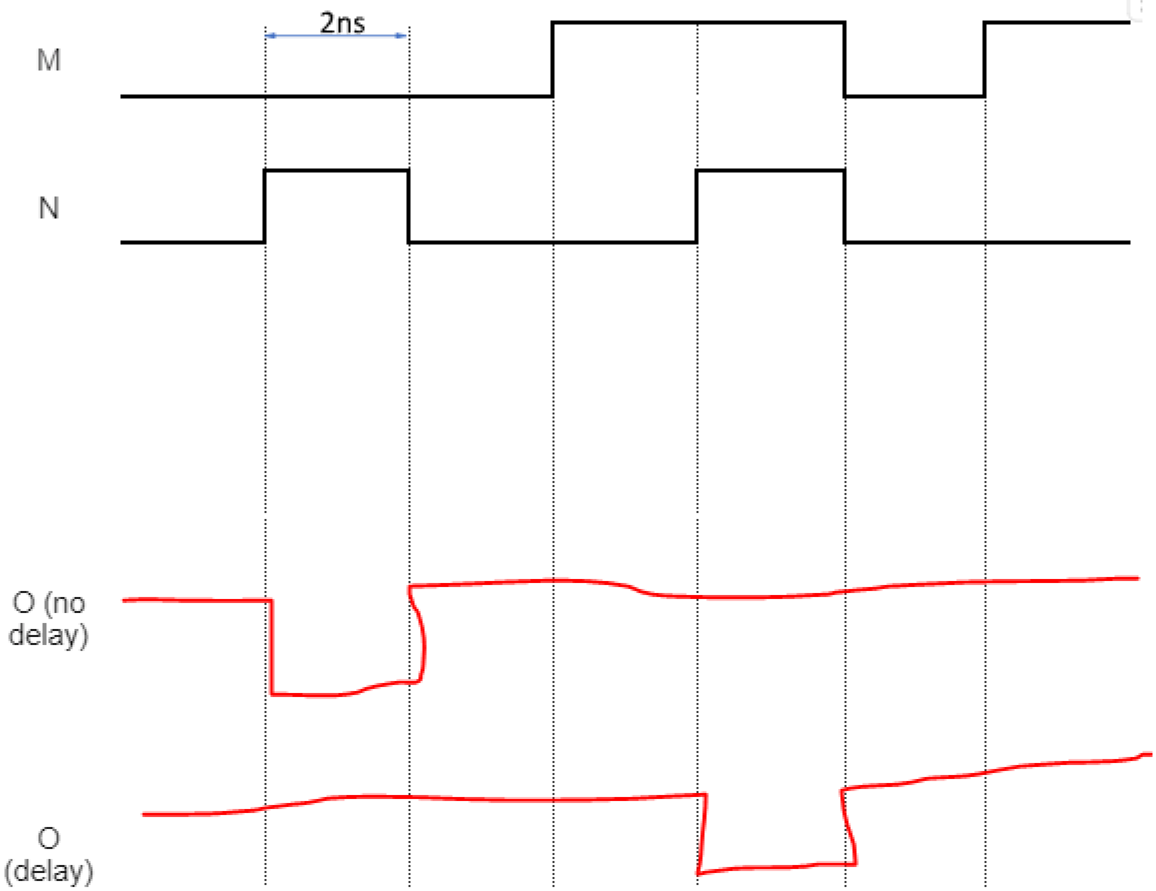
What is the **largest number that's still smaller** than 4.0 that can be encoded exactly in VSF (specify your answer in decimal)? *[3 points]*

4 - .0625 * 2 = 3.875

3. Consider the following circuit

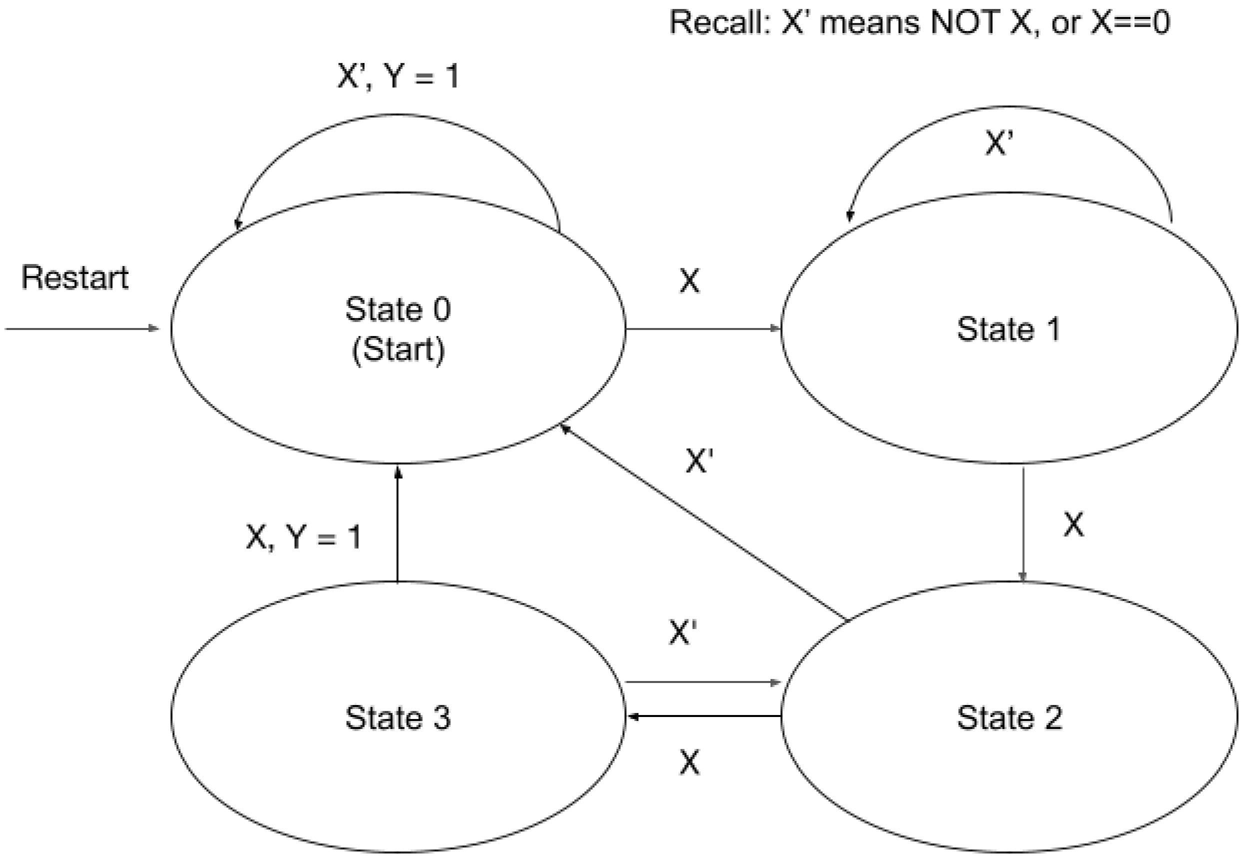


Complete the timing diagram for the circuit above **twice**, once with no delay, and one where each NOR gate has 2 ns of delay. Only represent the output current for the following 14ns window. Assume the starting values for M and N did not change recently before the diagram. You may draw intermediate signals, but these will not be graded. *[4 points]*



## Problem 3: Finite State Machines 10 points

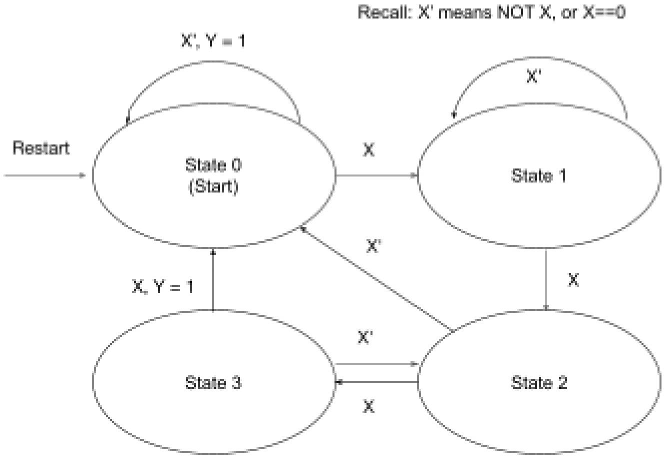
Consider the FSM below, with a single input X and a single output Y. Y is only output as 1 for the two transitions shown, all other transitions output Y=0.



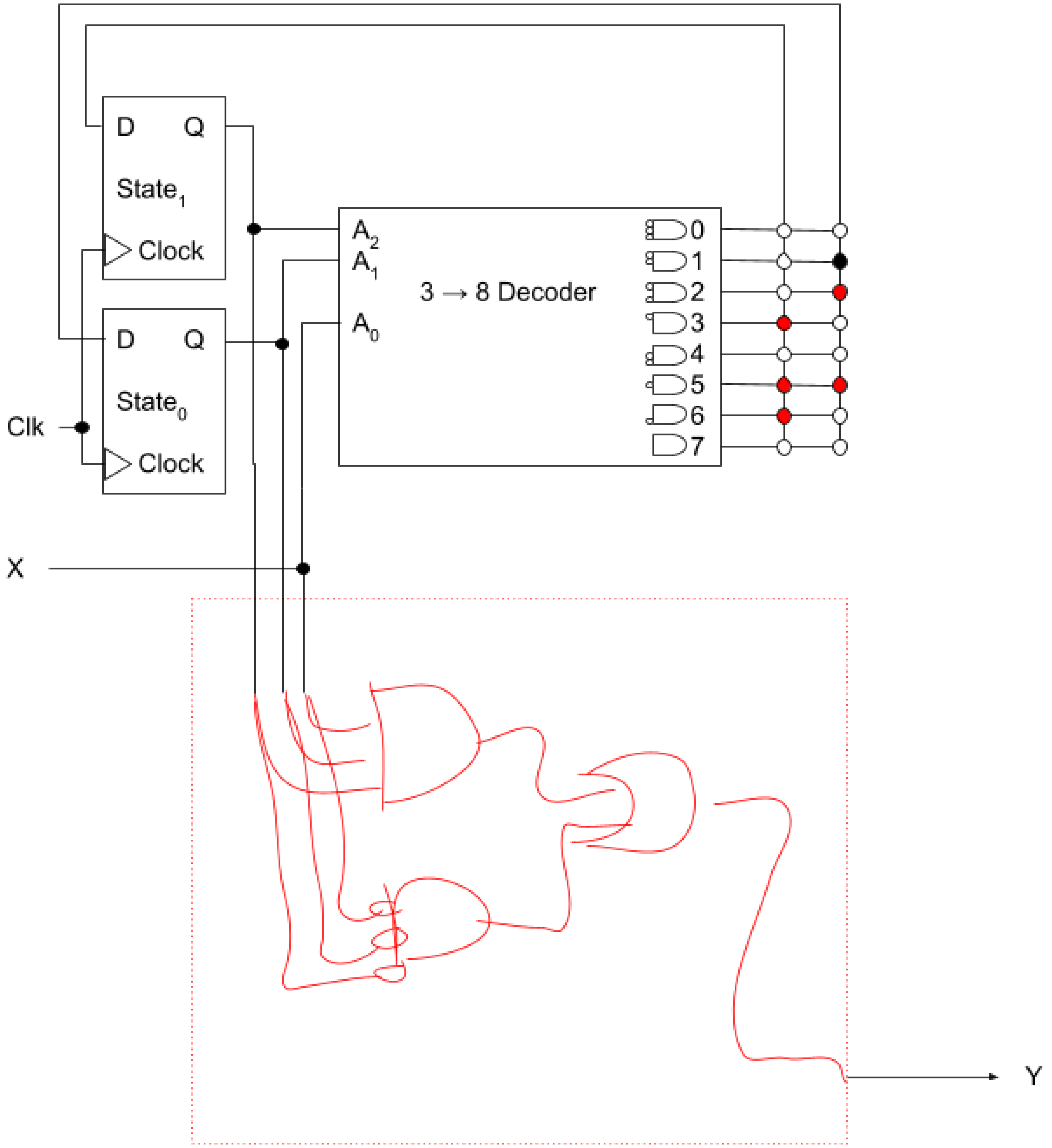
1. Is this a Mealy machine or a Moore machine? ____Mealy________ [1 point]

2. On the next page, convert the Finite State Machine into a circuit. Fill in the provided bubbles in the control ROM to determine the next state. Inside the dotted box, add wires and logic gates (using AND, OR and NOT - 3 input versions are fine) to complete the rest of the logic for the output Y *[9 points]*.
   - Assume the clock is connected but not shown.
   - Assume the flip flops have a starting value of 0.
   - Do not add outputs to the control ROM, and do not modify anything outside the ROM bubbles and the dotted box.
   - Filling in one of the control ROM bubbles means there is a connection, so current flows and we get a 1. Otherwise, the wires are disconnected and we get a 0.
     - For example, state 0 is done for you in the first 2 rows. When the FSM is in state 0 and X=0, the next state is 0, so both Next State bits are 0 and not filled in. But when the FSM is in state 0 and X=1, the next state is 1 (binary 01), so the bubble for the Next State$_0$ bit is filled in while the bubble for Next State$_1$ is not.

For your convenience, here is the unmodified FSM again:



**FSM Circuit** (reminder: AND, OR, and NOT gates only - 3 input versions are fine)



## Problem 4: Linking 8 points

Complete the symbol and relocation tables for the following C program (recall that #defines are resolved in a pre-compiler phase):CLARIFICATION: Any of use of "pancackes" in "void cook" refer to the local variable, not the global variable with the same name

pancakes.c

```
 1.    extern void flip(char **,
                        int index);
 2.    #define MAX_PANCACKES 10
 3.    char pancakes[MAX_PANCACKES][10];
 4.
 5.    void serve() {return;}
 6.
 7.    void cook(char** pancakes,
 8.              int num_cakes)  {
 9.      static int func_calls = 0;
 10.     func_calls++;
 11.     if (num_cakes == MAX_PANCACKES )
 12.        serve();
 13.
 14.     for(int i=0; i < num_cakes; i++)
 15.        flip(pancakes, i);
 16.     return;
 17.   }
```

List either "**text**", "**data**", or "**undefined**" under "Section", and either "**load**", "**store**", or "**jump**" under "Inst Type". You may not need each line.

pancakes.o Symbol Table

| Symbol | Section |
|---|---|
| flip | undefined |
| pancakes | data |
| serve | text |
| cook | text |
| *func_calls* | *data* |
| | |

pancakes.o Relocation Table

| Line # | Instr Type | Symbol |
|---|---|---|
| 10 | load | func_calls |
| 10 | store | func_calls |
| 12 | jump | serve |
| 15 | jump | flip |
| | | |
| | | |

## Problem 5: Struct Alignment 8 points

1. For the struct definition below, fill in the table assuming a 64-bit byte-addressable architecture. Assume that the first byte allocated to *townsville* is located at memory address 1000 *(5 points)*

```
struct {
    char mojo[10];
    struct {
        char  blossom;
        int   bubbles;
        int*  buttercup;
    } powerpuff;
    char mayor[3];
} townsville;
```

| Identifier | | | Size (bytes) | Start Address (in decimal) | End Address (in decimal) |
|---|---|---|---|---|---|
| | mojo | | 10 | 1000 | 1009 |
| | | blossom | 1 | 1016 | 1016 |
| | | bubbles | 4 | 1020 | 1023 |
| | | buttercup | 8 | 1024 | 1031 |
| | powerpuff | | 16 | 1016 | 1031 |
| | mayor | | 3 | 1032 | 1034 |
| townsville | | | 40 | 1000 | 1039 |

2. Consider struct **`powerpuff`***.* What ordering would result in the **most** padding bytes possible for this struct? How many bytes of padding will the **`powerpuff`** struct have in this case? (Note: there may be several orderings which maximize padding - provide **one** of them) (3 points)

1st - ____blossom_____________

2nd - ___buttercup_____________

3rd - ____bubbles_____________

Padding Bytes: _____11________

## Problem 6: Caller-Callee 10 points

The code below uses two new-to-us ARM instructions, push and pop. The push instruction stores the register value listed onto the stack. The pop instruction loads the data from the stack and places it in the listed register. Each of these instructions increments or decrements the stack pointer (stored in X13 aka SP) as appropriate. Assume the compiler checks for liveness across function calls when inserting caller/callee saves, but makes no other optimizations.

```
main:
      mov r1, #0x11
      mov r2, #0x22
      mov r3, #0x33
      mov r4, #0x00
loop:
      cmpi r4, #0x4
      ~~b.gte~~ end (should be b.ge)
      push {r2}
      bl batman
      pop {r2}
      push {r2}
      bl batgirl
      pop {r2}
      lsl r3, r1, #3
      lsl r2, r2, #4
      addi r4, r4, #0x1
      b loop
batman:
      push {r4}
      mov r2, #0x10
      add r4, r2, r2
      pop {r4}
      br x30
batgirl:
      push {r1}
      movz r1, #0x40
      movz r2, #0x50
      add r3, r2, r1
      pop {r1}
      br x30
end:
```

1. Each register is being used as either callee or caller-saved. Based on their usage, identify what each register is being used as. Then, indicate the number of load/store **pairs** that occur when the program "main" is run.

| Register | Caller or Callee | # Load/store |
|---|---|---|
| r1 | callee | 4 |
| r2 | caller | 8 |
| r3 | caller | 0 |
| r4 | callee | 4 |

2. If we flipped which registers were used for caller vs callee saved, how many load/store **pairs** would occur for each register.

| Register | # Load/store |
|---|---|
| r1 | 8 |
| r2 | 8 |
| r3 | 4 |
| r4 | 8 |

## Problem 7: ARM Assembly 12 points

Fill in the blanks to complete the following C-to-LEGv8 assembly conversion. You may not define any other labels. Assume we are following the ARM ABI discussed in class (see reference material). Remember that LEGv8 is a 64-bit system.

```
typedef struct {
  char* name;
  unsigned short age;
  int weight;
} Dog;

int get_age_dog_years(Dog* dog) {
  unsigned short age = (*dog).age;
  int weight = (*dog).weight;
  if(weight < 20)
      return age * 7;
  else
      return age * 8;
}
```

```
get_age_dog_years:
        ___LDURH__ X19, [X0, __8____]

        ___LDURSW_ X20, [X0, __12___]

        ___CMP   X20, #20___________

        b.ge____________ .else

        LSL X0, X19, #3_______________

        sub X0, X0, X19

        BR X30__________________

.else
        LSL X0, X19 #3_______________

        BR X30
```

## Problem 8: Performance 7 points

Assume that processor Alpha ~~is able which~~ executes a program in 2 ms. Processor Beta ~~Now~~ is an implementation of the multi-cycle datapath discussed in class. The program consists of 1 million instructions with the following instruction breakdown:

- 25% `add`
- 10% `nor`
- 20% `lw`
- 30% `sw`
- 15% `beq`

1. What is the CPI of processor Beta? Put your final answer in the box (specified to the nearest hundredth) and show your work below. *[3 points]*

4.20

.2*5 + .8 * 4 = 4.2

In order to be faster than Processor Alpha, the clock period of the Processor Beta must be [circle one] ***greater than /*** [✓ ***less than***] ___0.48________ ns (show to the nearest hundredth of a nanosecond). Show your work below. *[4 points]*

4.20 * 10 ^ 6 * X <= 2*10^6 ns

X <= 2*10^6 ns / (4.2 * 10 ^ 6)

X <= 0.48 ns

## Problem 9: Multi-cycle Datapath 20 points

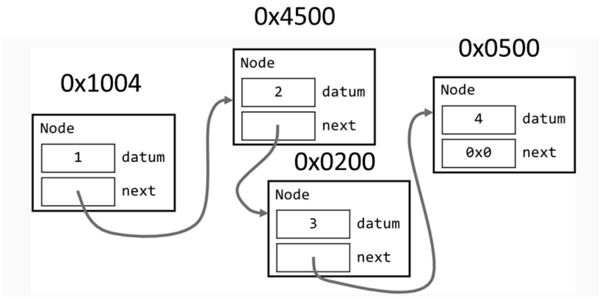
In homework 1, you saw how to write LC2K code to process a linked list. Your current employer, CISCyBusiness™, has set out to build a custom, more CISC-like LC2K processor to accelerate data structure processing. To that end, they are replacing the noop instruction with an I-type `findEnd` instruction,

```
findEnd  [regA]   [regB]   [offset]

REQUIRES:  (processor does not check for these)
           regA contains address of linked list with at least 2 nodes
           [offset] describes how many words into the node structure a
           pointer to the next Node is located
           regA index != regB index

EFFECTS:   Traverses linked data structure contained in regA until a
            null (zero) pointer is found. Address of the last node is
           stored in regB. No other registers updated.
```

For example, if memory is set up as below (with the Node struct shown on the right):



```
struct Node {
   int datum;
   struct Node *next;
};
```

and register 2 holds the address 0x1004, executing:

```
findEnd  2   1   #1
```

will write 0x0500 into register 1 (we use an offset of #1 here since the `next` field of the struct that we want to follow is located 1 word into the struct). **To avoid some edge cases, you can assume the list contains at least two nodes** (i.e. regA will never hold the address of the last node.)

Your employer wants to integrate this into our multi-cycle datapath following this procedure:

**Cycle 1**: Fetch instruction
**Cycle 2**: Decode instruction
**Cycle 3**: Calculate address of first "next" pointer
**Cycle 4**: Read pointer from memory
**Cycle 5**: Check if null pointer, if so instruction is finished
**Cycle 6**: Otherwise, update the result and calculate next address
[repeat earlier steps as needed until done]

1. On the following page is the multi-cycle datapath we will use to implement this. **Note that we have pulled out the "data register" which must be explicitly enabled to store a new value**, and we have connected the ALU equality check to the control ROM. No new gates are needed, but some signals may need to be routed as new inputs to the muxes [A-E]. List the extra connections needed to support these operations (do not include extra - unneeded connections will lose points). You do **not** need to verify the requirements (e.g. regA != regB). The first two states have been done for you. Leave unneeded lines blank. *[5 points]*

2. On page 16, describe the operations for each of the new states added in the finite state machine (recall that we used states 0-12 for the original multi-cycle design, so after decoding `findEnd` we move to state 13). You may only use the following components and syntax to describe the operations

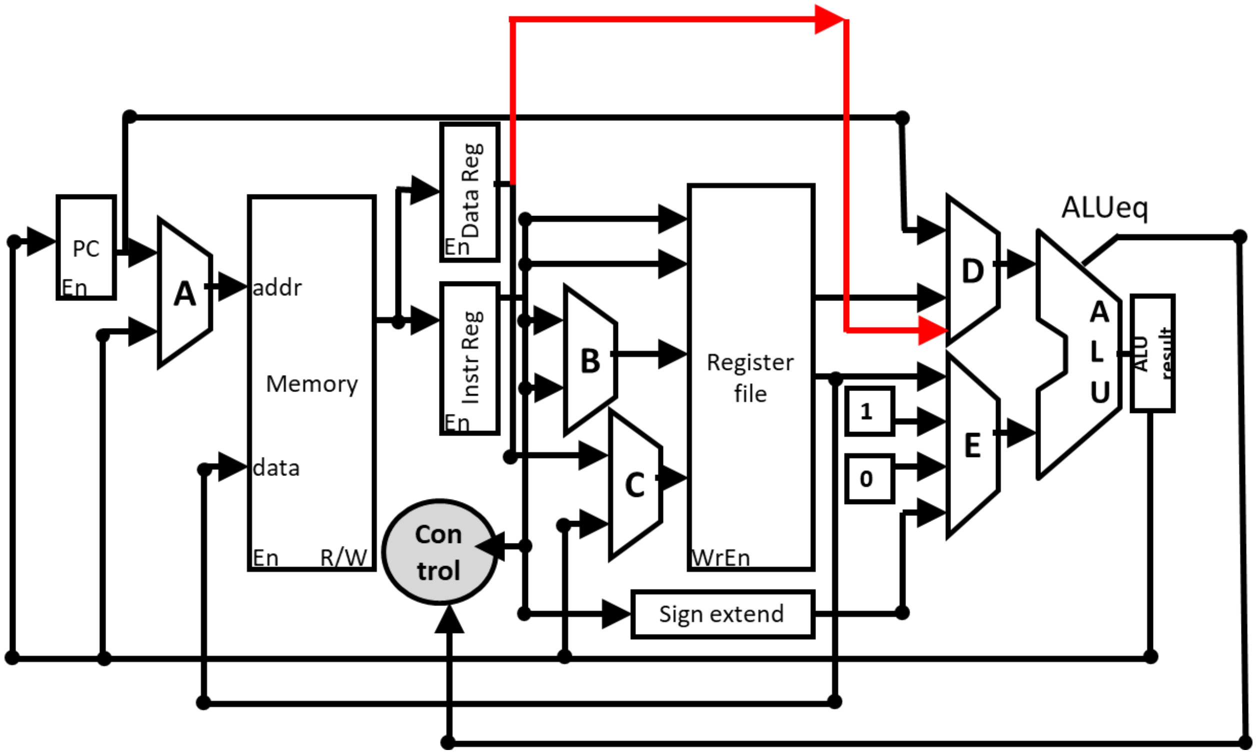
   ```
   MEM[___]
   regA, regB, offset, destReg, PC
   InstrReg, DataReg, ALUresult, ALUeq
   ```

   In addition, you must also indicate the next state logic for each state. If the next state is fixed, just indicate its number. For any state with conditional next states, (like the decode state 1), write a logical expression using ternary statements and signals passed into the controller to indicate how the next state is calculated. These ternary statements should have the format:

   ```
   [boolean expression] ? [state# if expression is true] : [state# if expression is false]
   ```

   The only signals the controller has access to are InstrReg[24:22] (i.e. the opcode) and ALUeq. *[15 points]*

### Part 1 - Modify datapath



New connections (leave unused lines blank):

Signal must be one of:

```
regA, regB, offset, destReg, PC
InstrReg, DataReg, ALUresult, ALUeq
```

Each mux must be a letter A through E.

Signal __DataReg______ is added as input to mux ___D____

Signal ________________ is added as input to mux ________

Signal ________________ is added as input to mux ________

Signal ________________ is added as input to mux ________

**Part 2 -**

| **State - Description** | **Actions** | **Reminder:** The only components you may reference are:<br>● `MEM[___]`<br>● `regA`<br>● `regB`<br>● `offset`<br>● `destReg`<br>● `PC`<br>● `InstrReg`<br>● `DataReg`<br>● `ALUresult`<br>● `ALUeq`<br>For next state specifically, you may only reference these as inputs:<br>● InstrReg[24:22]<br>● ALUeq*<br>*Hint: unlike the other signals listed, ALUeq is **not** a register output |
|---|---|---|
| 0 - Fetch | InstrReg = MEM[PC]<br>ALUresult = PC + 1<br>next state = 1 | |
| 1 - Decode | PC = ALUresult<br>next state = (InstrReg[24:22] == ADD) ? 2 :<br>***[other opcodes omitted]***<br>(InstrReg[24:22] == findEnd) ? 13 | |
| 13 - Calculate starting address | __ALUresult = regA + offset_____________<br>___________________________________<br>next state = __14____________________________ | |
| 14 - Load next pointer | ___DataReg = MEM[ALUresult]____________<br>_________________________________ __<br>next state = __15____________________________ | |
| 15 - Check if null pointer | ___ALUeq = (DataReg == 0)________________<br>___________________________________<br>next state = __(ALUeq == 1) ? 0 : 16__ | |
| 16 - Calculate next address and update result | __ALUresult = DataReg + offset___________<br>__regB = DataReg____________________<br>next state = ___14__________________________ | |

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK