GOOD LUCK TO EVERYONE!

So soon before the final, I have disabled editing and enabled commenting/viewing only.

cTHANK YOU! ;)

ADDED SOME DATA PATHS NEAR THE END. Hope that helps.

1. What is the decimal equivalent of the IEEE-754 floating point number:
1 10000000 10010000000000000000000
We have to subtract 127 from it to get the number of times the original binary number was shifted based on it's power. So 128-127 = 1. Now, we just shift the exponent to the right once. So we got from 1.1001 to 11.001, ignore the 0's in the exponent when shifting.
]
We now have 11.001
So we turn the first 11 into dec which is 3.
Than we turn the last 001 into a 1 in dec however we have to divide that 1 by half a certain number of times based on the bits there are in the fraction part, in this case 3: 001
So ½ = .5 -> .5/2 = .25 -> .25/2 = .125 We divided 3 times

So the answer is -3.125

2. Convert the decimal number #-476 to a 12-bit two's complement binary number, and represent the result as hexadecimal
a. x1BC b. x1BD c. x1DC d. xE23 e. xE24 f. xE44

First, represent +476 as a 12-bit two's complement binary number:
(ex. 0000 0000 0000 corresponds to 0 * 2^11 + 0 * 2^10 + 0 * 2^9 + 0 * 2^8 + 0 * 2^7 + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 0 * 2^0)
Now for 476:

```
2^11 = 2048   > 476 -> 0 * 2^11
2^10 = 1024   > 476 -> 0 * 2^10
2^9  =   578  > 476 -> 0 * 2^9
2^8  =   256  < 476 -> 1 * 2^8
476  -   256   = 220
2^7  =   128  < 220 -> 1 * 2^7
220  -  128    =  92
2^6  =    64  <  92 -> 1 * 2^6
92   -    64   =  28
2^5  =    32   > 28 -> 0  * 2^5
2^4  =    16   < 28 -> 1  * 2^4
28   -    16    = 12
```

```
2^3  =    8   < 12 -> 1  * 2^3
12   -    8   =  4
2^2  =    4   =  4  -> 1  * 2^2
4    -    4   =  0
2^1  =    2   >  0  -> 0  * 2^1
2^0  =    1   >  0  -> 0  * 2^0
```

So +476 as a 12-bit binary number: 0001 1101 1100
Two's Complement 12-bit binary number: NOT +476, then add +1:
1110 0010 0011 + 0000 0000 0001 = 1110 0010 0100

(finally!) convert to hexadecimal:
1110 = 2^3 + 2^2 + 2^1 + 0*2^0 = 14 = E
0010 = 0*2^3 + 0*2^2 + 2^1 + 0*2^0 = 2
0100 = 0*2^3 + 2^2 + 0*2^1 + 0*2^0 = 4

So the answer is xE24

3. Given the instruction (located at address x3500)
x3500 LDI R1, label1
and given:
label1 translates to address x35A0, which contains the value xC000;
memory location xC000 contains the value xD000;
memory location xD000 contains the value xFFFF;

What value will R1 contain after the instruction executes?
a. x3500
b. xC000
c. 35A0
d. xD000
e. xFFFF
LDI is essentially a pointer to a pointer to a value. That is, label 1 is at the address x35A0 and holds xC000, which is then used as an address to find the value, xD000.
Mem[Mem[Label]] <- short explanation of ldi
Mem[Mem[x35A0]] = Mem[xC000] = xD000

4. Simplify the Boolean expression:
a.b'.c'.d + a.b'.c.d + a.b.c'.d + a.b.c.d' + a.b.c.d
a. a.b.c + a.b.d + b.c.d
b. a.d + a.b.c
c. a.c + a.b.d
d. a.c.d + a.b
e. a.c + b.d + b.c

f. can’t be simplified
a.b’.c’.d + a.b’.c.d = a.b’.d
a.b.c’.d + a.b.c.d = a.b.d
a.b.c.d’ + a.b.c.d = a.b.c
a.b’.d + a.b.d = a.d

a.d + a.b.c
Remembering that we can “or” an extra term
a.b.c.d without altering the value (since x + x = x)
this is what the tutorial states

5. How many memory locations can be addressed by a microprocessor that uses 24 bit addressing?
2^24 = 2^4 * 2^20 = 16 M
a. 16 k
b. 512 k
c. 16 M
d. 2 G
e. 4 G
f. none of the above

6. How many select lines does an 8 input multiplexer have?
2^3 = 8, so 3 select lines
a. 1 b. 2 c. 3 d. 8 e. 64 f. 256

7. How many input lines does an 8 input multiplexer have?
8 inputs lines...because theres 8 inputs. Weird question
a. 1 b. 2 c. 3 d. 8 e. 64 f. 256

8. How many output lines does an 8 input multiplexer have?
Remember multiplexers only have 1 output!
a. 1 b. 2 c. 3 d. 8 e. 64 f. 256

9. How many inputs does a full adder circuit have?
Full adder has 3 inputs A, B, and the Carry
a. 1 b. 2 c. 3 d. 4
e. It depends on the number of bits in the numbers being added
Remember that a full adder circuit has 2 outputs - don’t get tricked!

10. A “gate delay” can be described as the time needed for the output of a gate to settle

to its correct level after one of its inputs has been changed. The full-adder circuit we have designed would therefore result in a gate delay of 2 units.
How many units of gate delay would a 4-bit ripple-adder display?
a. 2 b. 4 c. 8 d. 16 e. 32

For each bit of the ripple-adder display, there is a circuit that carries the um, electric signal? through it. And for each circuit that it is carried through (4 circuits in this case), there is a gate delay of n units (2 in this case). 2 + 2 + 2 + 2 or 2 * 4 = 8
Juan told me this, I have (misguided?) faith in him!

11. How do we turn 8 separate gated D-latches into a single 8-bit register?
(WE = "Write Enable")

a. connect the WE of each one separately to an output of a 3-bit decoder, and multiplex their outputs with
the same decoder
b. connect the inputs of each one separately to an output of a 3-bit decoder, and multiplex their WEs with the same decoder
c. Use a 3-bit decoder as a selector, multiplex their WEs
d. connect their WE lines together to a single external WE
e. super-glue them together

12. How do we turn 8 separate gated D-latches into 8 addressable registers, each 1-bit wide?
a. connect the WE of each one separately to an output of a 3-bit decoder, and multiplex their outputs with the same decoder
b. connect the inputs of each one separately to an output of a 3-bit decoder, and multiplex their WEs with the same decoder
c. Use a 3-bit decoder as a selector, multiplex their WEs
d. connect their WE lines together to a single external WE
e. super-glue them together

13. The following very complex subroutine moves the cursor down to the next line.
What, if anything, is wrong with it?

```
.ORIG x4000
NEWLINE LD R0, CRLF
OUT
RET
CRLF .FILL x0A
```

Need to backup the registers you use, in this case, R0 because it is changed and always R7
a. Nothing - it's fine as it is

b. x4000 is not a valid starting address for a subroutine
c. it should save and restore the contents of R0
d. it should save and restore the contents of R7
e. both c) and d)

14. A subroutine was called using a conditional branch (BR) instruction. The subroutine ends, as usual, with a RET instruction. What will happen when the subroutine terminates?
a. Control will return to the original BR instruction
b. Control will return to the instruction following the BR instruction
c. Control will return to an unknown instruction, either crashing the program or producing unpredictable results.
d. It will depend on which of the NZP condition codes the BR instruction tested

The problem here is that BR does not store the original PC in R7 like JSR or JSRR would. Instead it overwrites the PC with the label in the instruction if the conditions meet and jumps there. So when RET decides to try to jump back to an address stored in R7, it will cause unpredictable results because the original location was never saved properly before in R7 so R7 holds 'junk'.

The next three questions refer to the following scenario:
Consider the controller of an elevator that connects the 1st, 2nd, 3rd & 4th floors of a building as an example of a finite state machine.
The states of this FSM correspond to the elevator stopped at each of the four floors, either with the doors closed, or with the doors open (i.e. "doors open" and "doors closed" are separate states).
Transitions between states will thus correspond to the elevator moving between floors, including "express" transitions between non-adjacent floors, and opening and closing doors while stopped at any given floor (obviously, the elevator can move only if the doors are closed)
External inputs are determined by the elevator call buttons.

15. How many separate states does this finite state machine have?
Since theres 4 floors, and each has 2 states (open or closed), 4 * 2 = 8 total seperate states
a. 4
b. 6
c. 8
d. 12
e. 16
f. 32

16. How many distinct transitions are there between states (not including "null transitions", i.e. transitions from a state back to itself)?
Since we are not counting the NULL transition we will have 3 transitions to other floors instead of

4 because each floor can only go to other floors not the current floor it is at. So 4 floors and 3 transitions each. 4 * 3 = 12. Plus the 8 total seperate states. 12 + 8 = 20 distinct transitions

a. 8
b. 12
c. 16
d. 20
e. 24
f. insufficient information

17. How many bits would the Storage Logic component of the fsm have to store?

a. 1
b. 2
c. 3
d. 4
e. 5
f. huh??

we need 3 bits to represent 8

18. A computer system has a word addressable memory. Each word is 32 bits (4 bytes) wide. There are 24 memory address lines. What is the maximum available system memory?

Address space * Adressability (in bytes) = TOTAL MEMORY
16M (Address Space) * 4bytes(Addressability)= 64 M bytes

a. 64k bytes
b. 16M bytes
c. 64M bytes
d. 256M bytes
e. 4G bytes
f. 16G bytes

19. How many data lines are required for the system in the previous question?

Data lines are pretty much the amount of lines that are needed for each bit in a word. In this case our word is 32 bits, so we will need 1 data line for each bit of the word.

a. 8
b. 16
c. 24
d. 26
e. 28
f. 32

I suppose if the addressability is 32 bits, then this system is a 32 bit system right? Then at maximum there would be 32 circuits? and 32 decoders etc. to do all the control and processing tasks in the computer system. Data lines I guess are referring to the necessary 32 lines under the hood that produces 1’s and 0’s for all the processes in the computer.

20. If the system in the previous questions were instead byte-addressable, and were to retain the same total number of bytes of memory, how many address lines would it require?
Before it was:
4byte (addressability) * 16M (address space) = 64 Mbytes total available memory

But now with byte addressability:
1byte (addressability) * x (address space) = 64 Mbytes total available memory
x (address space) = 64 M
64M (address space) = 2^20+2^6= 2^26
a. 8
b. 16
c. 24
d. 26
e. 28
f. 32

21. Given a very peculiar memory system that uses 22 bit-addressing, and is "three-bit" addressable (i.e. each location in memory stores 3 bits), how many bits of storage does the memory contain in total?

a. 88 bits
b. 64kbits
c. 1 Mbits
d. 12 Mbits
e. 16 Mbits
f. 64 Mbits

22 bits = 4M * 3 bits = 12 Mbits
The prefix M takes on the unit Mbits instead of Mbytes when you multiply by 3 bits.

22. The central idea in the von Neumann model is that the program and data both reside in:
More of a definition question

a. multiplexers
b. decoders
c. adders
d. memory
e. switches

f. models

The next three questions refer to the following system:
A certain ISA has a 32-bit word size, uses single word (32-bit) instructions, has 60 opcodes, 32 registers, and 4Gbyte of byte-addressable memory.
One group of instructions in this ISA takes the form:
OPCODE | DESTINATION REGISTER | SOURCE REG. | Flag | IMMEDIATE VALUE
Or
OPCODE | DESTINATION REGISTER | SOURCE REG. 1 | Flag | SOURCE REG. 2
A single bit in the instruction ("Flag") is used to differentiate these two addressing modes.
Another group of instructions takes the form
OPCODE | SOURCE/DESTINATION REGISTER | PC OFFSET
Where PC Offset is the 2's complement "distance" from the current PC to the labelled location.

60 Opcodes = 6 bits
32 Registers = 5 bits

23. What is the range of values that can be stored in the "Immediate" field (as a 2's complement value)?
Use
OPCODE | DESTINATION REGISTER | SOURCE REG. | Flag | IMMEDIATE VALUE
32(word size) - 6 (opcode) - 5 (Register) - 5(Register) - 1(Flag) = 15 bits for the immediate
2^15 = 32 K = (-16) to + (16k-1)
a. -16 to +15
b. 0 to (64k – 1)
c. –16k to +(16k – 1)
d. –32k to +(32k – 1)
e. –64k to +(64k – 1)

24. How many bits are required for addressing (i.e. what is the size in bits of an address)?
a. 16
b. 24
c. 32
d. 48
e. 4G
4G = 2^32 so 32 bits
poop

25. How far (in memory locations) can the label be from an instruction that references the label, using the <opcode | register | pc offset> addressing mode?
You may assume that in this assembly language, each line of source code assembles to a

single one-word (32-bit) instruction.

OPCODE | SOURCE/DESTINATION REGISTER | PC OFFSET
32(word size) - 6(opcode) - 5(Reg) = 21
21 bits = 2M = +/- 1M

a. + / - 32k
b. + / - 64k
c. + / - 128k
d. + / - 256k
e. + / - 512k
f. + / - 1M

26. An ISA specifies a word size of 8 bytes; word addressability; and an address space of 4G; it uses single-word instructions (i.e. each instruction is a single 8 byte word). What are the sizes of the PC and the IR?
a. both 24 bits
b. both 32 bits
c. both 64 bits
d. PC: 24 bits; IR: 64 bits
e. PC: 64 bits; IR: 32 bits
f. PC: 32 bits; IR: 64 bits

27. For the system in the previous question: What are the sizes of the MDR and the MAR?
a. both 24 bits
b. both 32 bits
c. both 64 bits
d. MAR: 32 bits; MDR: 64 bits
e. MAR: 64 bits; MDR: 32 bits
f. MAR: 24 bits; MDR: 64 bits

Recall the Fetch microinstructions in Ch4 Slide # 14 or #31 of this pdf:
Both the MAR and the PC have to do with address space, which is 4G = 2^32 = 32 bits.
Both the MDR and the IR have to do with the instructions and word size, which is 8 bytes = 32 bits. Since LC-3 uses 2 word instructions, you need 2 * 32 bits = 64 bits. I'm 100% positive that this is true, but can someone back it up with some sort of proper explanation? lol

^^We are not talking about the LC-3 in these series of questions. The ISA was defined with an 8 byte word. This means that a single word is 8 * 8 bits = 64 bits. So what this means is that there are 4G locations and each location points to 64 bits. Now think of a label for this machine; a label names an address location and so what can the label store? The label's data would be limited to the word size. Thus we are dealing with Data that is 64 bits long and the MDR should be that size.

^i'm glad people are still awake

28. Consider the following LC-3 code fragment (the hex values in the first column give the address to which the corresponding instruction is loaded):
x34FE loop1 ADD R4, R5, R6
.....
x35A0 BRnp loop1 ; NOTE THE TEST IS "np"
Given: the BR opcode is 0000 NZP
What does the instruction at x35A0 assemble to? (Remember to account for the current value of the PC when the BR instruction is being executed, and the direction of the "jump", and the condition codes to be tested)
a. x0B5D
b. x0B5E
c. x0B5F
d. x0AA1
e. x0AA2
f. x0AA3

This question is basically asking for the hexadecimal representation for the 16 bit binary instruction BR(np) from a location x35A0 to location x34FE. To do this, one can remember from class or from the end of the lab 08 pdf that the binary instruction for BR looks like:
0000 101 xxxxxxxxx
opcode nzp PC Offset 9

The opcode is the 4 bit code corresponding to which instruction (eg. ST, LD, BR, Trap) the binary is referring to. In this case, 0000 is the opcode for BR.

The three bits denoting nzp indicate which cases are being tested for; in this example n and p are being tested for, so those bits are 1's while the z bit is a 0.

PC Offset 9 means the 9 bit 2's complement pc offset value to the label. The way PC offset works is you find the distance between the location of PC (the line after the instruction line you are looking at), and the address of the memory you are trying to branch to. x35A0 is the location of the BR instruction so the location of the PC is x35A1.
The distance for the PC Offset is x35A1 - x34FE = x00A3

Let me explain this subtraction real quick:
x35A1 - x34FE if you were to try to subtract it the way you do with decimal numbers, but using a base of 16 instead of 10, you would need to borrow digits, just like with regular subtraction, right? So after borrowing digits the digits/subtraction would look like this:
x3|4|(19)|(11) in hexadecimal

-x3|4| F | E

____

x0|0 | A | 3
= x00A3

In this case, BR is jumping to a location backwards in memory, so the PC offset is considered negative. This means we have to do the two's complement of xA3 before we can use it in the binary form [opcode] [n][z][p] [PC Offset 9].

x00A3 = xA3 (can drop the first 2 hexadecimal digits - they have to be 0 anyways b/c the PC offset is +/- 2^8, any more would cause overflow!)

xA3 = 0 1010 0011 -> NOT -> 1 0101 1100 -> ADD x1 = 1 0101 1101 = x5D
opcode nzp PCOffset9
So this BR instruction has the binary form 0000 101 1 0101 1101 which in hexadecimal is: x0B5D

29. How many separate decoders are there for the general purpose register bank in the LC-3?
a. 1 b. 2 c. 3 d. 4 e. 8
f. The register bank does not require decoders
8 registers - need 3 bits for 2^3 registers. Each bit uses 1 decoder for addressing.

30. In the Indirect mode of memory addressing in the LC-3 (e.g. the instructions LDI and STI), the Effective Address is calculated by:
Remember its a pointer so Mem[PC + SEXT(IR[8:0])]
a. (BaseReg) + SEXT(IR[5:0])
b. (PC) + SEXT(IR[8:0])
c. Mem[(PC) + SEXT(IR[5:0])]
d. Mem[(PC) + SEXT(IR[8:0])]
e. (IR) + SEXT(PC[5:0])
f. Mem[(IR) + SEXT(PC[8:0])]

31. What are the "micro-instructions" that comprise the Fetch phase of the Instruction cycle?
a. IR <- (PC); MAR <- (IR); PC <- Mem[MDR]; PC <- (PC) + 1;
b. PC <- (MDR) + 1; MAR <- Mem[PC]; IR <- (MDR)
c. MAR <- (PC) + 1; MDR <- Mem[IR]; IR <- (MAR);
d. MAR <- (PC); PC <- (PC) + 1; MDR <- Mem[MAR]; IR <- (MDR);
e. MDR <- (PC); PC <- (PC) + 1; MAR <- Mem[MDR]; IR <- (PC)

32. How does the control unit decide whether to take the branch pointed to in a BR instruction?
(n, z & p represent IR [11:9]; N, Z & P represent the values of the condition code registers)
If you follow the data path of the BR instruction near the end of the data path when it LD.PC, it checks if(n.N + z.Z + p.P) = 1, and WE the PC
a. if (n . N + z . Z + p . P) = 1, the MAR is write enabled
b. if (n . N + z . Z + p . P) = 1, the PC is write enabled
c. if (n . N + z . Z + p . P) = 1, the EA corresponding to the label is calculated
d. if ((n + N) . (z + Z) . (p + P)) = 1, the MAR is write enabled
e. if ((n + N) . (z + Z) . (p + P)) = 1, the PC is write enabled
f. if ((n + N) . (z + Z) . (p + P)) = 1, the EA corresponding to the label is calculated

33. All control instructions in the LC-3 have one main step in common:
a. They all reconstruct the required memory address in the same way
b. They all use the ALU in reconstructing the required memory address
c. They all write to the IR in the execution phase of the instruction cycle
d. They all write to the PC in the execution phase of the instruction cycle
e. They all write to the MDR in the execution phase of the instruction cycle
f. They all write to the GPR bank in the execution phase of the instruction cycle
Control instructions always change the PC

34. The LC-3 instruction cycle consists of 6 phases. What does this mean?
a. 6 instructions require 1 cycle, while the other instructions may require more
b. Each instruction consists of up to 6 steps
c. The execute stage has 6 possible operations
d. 6 of the 16 bits of an instruction are dedicated to opcode
e. The processor has 6 main parts, including the ALU, register file, etc.

35. One of the four control signals to the LC-3 ALU is "pass-through input A" - i.e. input A is connected directly to the output.
Which of the following instructions would use this control signal?
ST, STI, and STR all pass through the ALUK according to their data paths
a. NOT
b. LD, LDI & LDI
c. ST, STI & STR
d. BR & JMP
e. JSR/JSRR
f. TRAP

36. In the LC-3, the DR decoder input comes from the DRMUX. What are two of the

inputs to the DRMUX? (Hint: think about the JSR and TRAP instructions)
Was one of the questions in our quiz. Also in JSR it selects the [111] but I'm not sure why...

JSR selects [111] because that is the binary representation of R7. I remember the professor saying that R7 is hardcoded as one of the inputs to the DRMUX to hold the "return address". That is why if R7 is not backed up when you call JSR, your subroutine does not know where to return to. If the instruction TRAP is called and you happen to be using R7 in your calculations, you will notice that R7 will be overwritten as soon as the Trap instruction is executed. That is why I think, in the very beginning of the quarter it was advised that R7 is left alone. Hope this helps.

a. IR[2:0], IR[86] and IR[11:9]
b. IR[8:6] and IR[11:9]
c. [111] and IR[11:9]
d. The system bus and IR[11:9]
e. (PC) and IR[11:9]
f. PC[2:0] and IR[11:9]

37. In the LC-3, the SR1 decoder input comes from the SR1MUX. What are two of the inputs to the SR1MUX? (Hint: think about the load and store instructions)
a. IR[2:0], IR[8:6] and IR[11:9]
b. IR[8:6] and IR[11:9]
c. [111] and IR[11:9]
d. The system bus and IR[11:9]
e. (PC) and IR[11:9]
f. PC[2:0] and IR[11:9]
WHY??
ADD/AND/NOT/LD/LDI/LDR have SR1 in [8:6] and ST/STI/STR have SR1 in [11:9]
See Lab08 (very end) and that should help

38. In the LC-3 (and most ISAs), the System Control Block, or Trap Vector Table, contains:
a. the complete Trap Service Routines
b. the 9-bit PC offsets of the Trap Service Routine addresses
c. the 8-bit entry point into the Trap Vector Table
d. the starting addresses of the Trap Service Routines
e. the return addresses to be used after returning from a Trap Service Routine

39. What is the main purpose of the first pass of a two-pass assembler?
a. to determine if the code will fit into available memory
b. to produce the machine language equivalent of the assembly language instructions

c. to link other possible object files in order to create the executable
d. to remove all pseudo-ops from the code before it is assembled
e. to build a symbol table relating labels to memory addresses

Can anyone tell me the second pass?
^this was also me

40. In a cpu that uses the technique of memory mapping to address ports (registers that interface between the cpu and peripherals), how must the ports actually be accessed?
a. via dedicated i/o instructions.
b. via interrupts
c. via polling
d. via standard Load/Store instructions
e. either a) or d)
f. none of the above

41. What component of the cpu gets "interrupted" by an Interrupt signal? (i.e. where does the Interrupt signal go to?)
a. The PCt
b. The PC-MUX
c. The MAR-MUX
d. The global bus
e. The FSM
f. The Memory Mapping logic

42. In Interrupt processing, what is the purpose of the IACK signal?<--Where is this in the slides/book?
a. It alerts a collection of peripherals that the cpu is available and will service interrupts
b. It polls multiple peripherals to find which initiated an interrupt
c. It is used by the cpu to flag to a peripheral that it has completed servicing its interrupt
d. It is used by a peripheral to flag to to the cpu that it no longer requires servicing
e. It is used by a peripheral to "lock" the bus once its interrupt has been acknowledged by the cpu

43. In the LC-3, the TRAP instruction and the interrupt handler both manage the invocation of service routines in a similar fashion. Specifically, both use:
a. polling of status registers to decide when to read from/write to a port
b. a trap/interrupt vector as an entry point into a table of service routine addresses
c. an IACK signal to determine which service routine is requested
d. a stack to store information required for the return, allowing nested calls
e. a system of task priority comparisons to determine whether to invoke the service routine
(See Page 260, last paragraph)
44. What system state information has to be saved before an interrupt-enabled LC-3 can

proceed with servicing an interrupt?
a. the value of every control signal produced by the finite state machine
b. the value of every control signal produced by the finite state machine, plus the contents of all Registers
(GPRs, PC, condition codes, etc.) – except the IR
c. the PC
d. the PC and all the General Purpose Registers
e. the PC, and the PSR (Processor Status Register, containing the NZP condition codes, the Privilege level, and the current task priority) I just memorized this.
f. the PC and the MCR (Machine Control Register)

45. At what point in the instruction cycle is an interrupt handled?
a. at any time during the entire cycle - the interrupt is just one of several external inputs to the FSM
b. at any time during the fetch instruction phase
c. only at the very start of the fetch instruction phase
d. at any time during the last phase (store)

46. The data protocol of a stack data structure is known as
Useful in nested loops
a. LIFO (Last In, First Out)
b. FIFO (First In, First Out)

47. The two main approaches to converting a HLL source code to ML (Machine Language) are:
a. direct and indirect
b. memory mapping and polling vergas
c. assembly and disassembly
d. interpreting and compiling Just looked it up.
e. compiling and linking

48. The structure which allows Higher Level Languages to make nested function calls is:
a. Activation records stored on the run-time stack (From CS 12)
b. The symbol table
c. The frame pointer
d. The Processor Status Register
e. The Machine Control Register

49. Which LC-3 assembly language instruction is most likely to be used to compile a HLL (Higher Level Language) access to a local variable:
LDR because it is used more like a pointer in C++. Correct me if I am wrong. You are right.
a. LDR b. LDI c. LD d. TRAP e. JSR

50. Which register is most likely to be used as the Base Register in accessing the variable, in the situation described in the previous question?

a. Frame pointer (R5 in the LC3)   I just memorized this. He mentioned this in class on thur

b. Top of Stack (R6 in the LC3)

c. Program Counter (R7 in the LC3)

d. Processor Status Register

^i uh have no idea what a frame pointer is. anyone?

Section II: Written answers

1. 15 points

Design a digital combinational logic circuit with four inputs: a, b, c & d, where (a, b) represents one 2-bit unsigned binary number A{1:0]; and (c, d) represents another 2-bit unsigned binary number B[1:0] (i.e. both A and B are in the range 0 to 3).

The circuit has 4 outputs (or you can regard it as being 4 distinct circuits, each with a single bit output) – in other words, the truth table will have 4 input columns and 4 output columns. These output columns together repesent the 4-bit **product** Y[3:0]

Y = A * B

For instance, inputs corresponding to "3 , 2" would output bits corresponding to 6

- Start by drawing up the truth table (6 points) (show **only** those rows which produce a 1 in any of the output columns)

    Make sure you label your input and output columns correctly – everything else depends on getting the table right!

- then derive the algebraic expression for the third bit of the output, Y[2] (3 points);
- and simplify it (3 points)
- Finally, draw the resulting circuit (3 points)

(Each part is *"all or nothing" - no partial credit*)

See problem 3.28 in book and the solution in HW4 on ilearn.

I think this is how you do it...
My truth table is really messy/unorganized but I think yall get the idea. So we have 2 bits from AB and 2 bits from CD. We want to multiply the 2 bits AB and CD and get the output Y. So in the first line of the truth table AB(1 in binary) * CD(1 in binary) = 0001 = 1 = Y. The second line is AB(2) * CD(2) = 0100 = 4 = Y. And so on.

Truth Table:

A	B		C	D	AB * CD	O_3	O_2	O_1	O_0	=	Y
0	1	|	0	1	|	0	0	0	1	=	1
1	0	|	0	1	|	0	0	1	0	=	2
1	0	|	1	0	|	0	1	0	0	=	4
1	1	|	1	0	|	0	1	1	0	=	6

This can go on longer, idk if linard wants us to put all possible combinations.

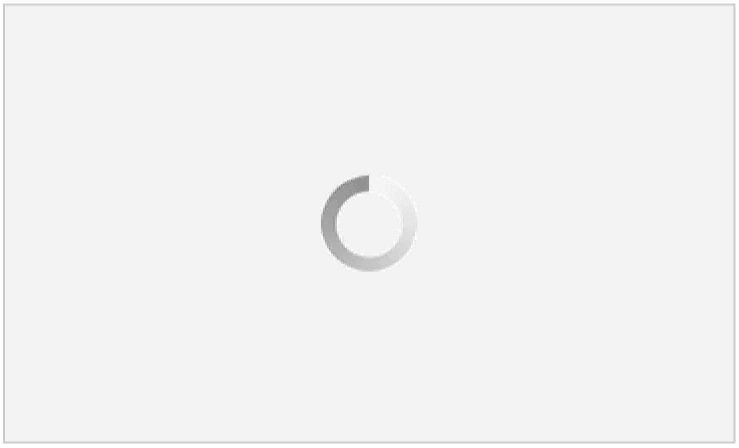
I think 4 and 6 are the only ones with 1 in the 3rd bit in the range of a 2bit product, so the boolean expression would be something like:

A.B’.C.D’ + A.B.C.D’

Simplified:

A.C.D’

This should be the circuit for bit 3



2. 15 points
Given the data path of the LC-3 (provided below), give a complete description of the STORE DIRECT instruction (ST SR, label), as follows:

a) Give the RT (Register Transfer) specification of the instruction (4 points)

b) List the data applied to all relevant circuits in the data path (i.e. just those circuits relevant to the ST instruction); and list, in the correct sequence, every control signal set by the FSM to implement this instruction. (11 points)

Remember that not every tri-state device may be actually depicted on the schematic – but you must still specify the control signal if it is involved (make up descriptive names for any control signals that are not shown on the schematic). Likewise, the SR MUX is not shown, but you must still include the correct select control signal and data input for it.
For multi-bit control signals, you must specify the actual value where possible:
e.g. if the control signal is a 2-bit MUX selector, selecting for the input that is third from the right, then you must specify the value 10

```
 |    |   |    |
_|____|___|____|_
|11  10  01  00|
|______________|
```

If several control signals act at the same time, indicate that fact; otherwise list them in sequence.

btw, this was a HW problem, on HW 7(not that anyone has done any of them)

a) ST RTN: Mem[(PC) + SEXT(IR[8:0])] ← SR

I'm not sure about which control signals are turned on or turned off in this case. If anybody could specify it, that would be great. This is just the general path of ST

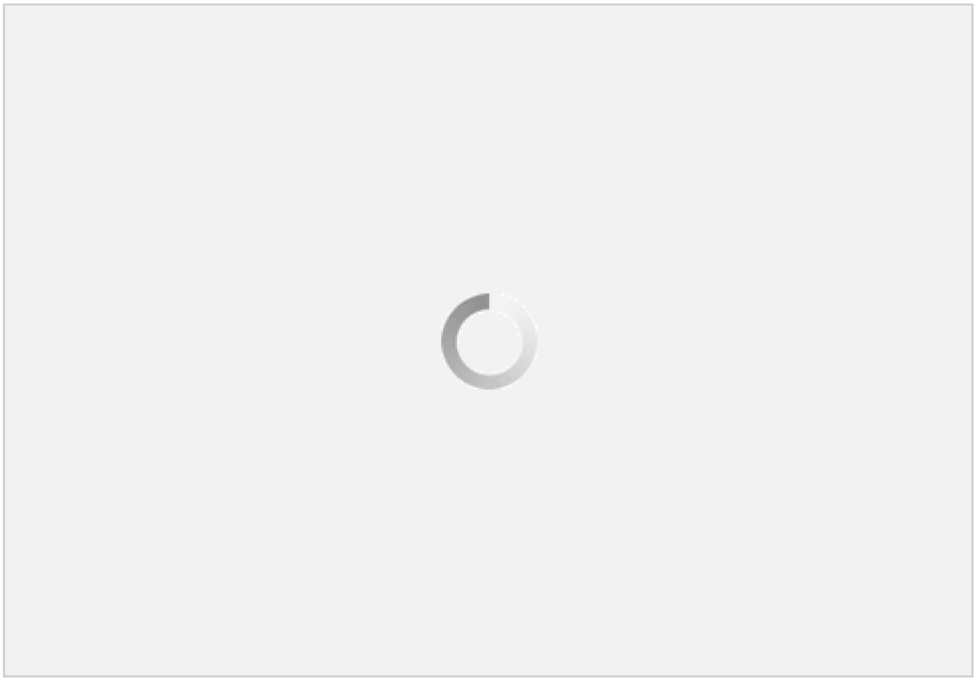
b) 1. Select SR1MUX: IR[11:9]
   2. ALU → Pass through ALUK
   3. Gate.ALU
   4. LD.MDR
  5. a. ADDR1 MUX: Select PC, selects bit 0
      b. ADDR2 MUX: Select SEXT(IR[8:0]), selects bit 10
  6. MARMUX, selects bit 0
  7. LD.MAR
  8. Memory: Memory enabled for writting

3. 15 points
Construct the Finite State Machine representation for a counter with a cycle length of 4 - i.e. a circuit that counts 0 – 1 – 2 – 3 (output as a binary value, obviously) with successive clock pulses, and then starts over.
The external output is the 2-bit count.
The only external input is R, a reset pulse: when R = 1 it resets the next count to 0, no matter what the current state; when R = 0 it keeps counting (i.e. the system transitions to the next state in sequence).

Then construct the complete truth table(s) for the device, showing
the inputs: "current state" labels (ie. the state we are transitiong from), and R
the outputs: "next state" labels (i.e. the state we are transitioning to), and the 2-bit count associated with that state.
(Hint: if you choose the state labels sensibly, they will be identical to the output)

Finally, derive and simplify the algebraic expression for bit 0 of the output.

I think for this problem it is much like the detour representation of the FSM we did in lecture, where we have the 4 circles each pointing to the next, something like this:



If R equals to 1 at any point the FSM goes to the starting point. If R just equals to 0 it continues counting.

Truth Table:

C is the current state, R is the input passed in, N is the next state, and Cnt is the 2bit count.

C_0	C_1	R	|	N_1	N_0	Cnt
0	0	0	|	0	1	1
0	0	1	|	0	0	0
0	1	0	|	1	0	2
0	1	1	|	0	0	0
1	0	0	|	1	1	3
1	0	1	|	0	0	0
1	1	0	|	0	0	0
1	1	1	|	0	0	0

You can see that whenever the R is 1, device resets back to 0, but if it is 0, the device continues counting unless it is on the last device where it would also reset.

Boolean expression for the 0th bit:

$C_0'.C_1'.R' + C_0.C_1'.R'$

Simplified:

$C_1'.R'$

If C1 and R are both 0, we know that the 0th bit will be 1

4. 5 points
A number of LC-3 instructions have an "evaluate address" step in the instruction cycle, in which a 16-bit address is constructed and written to the Memory Address Register via the MARMUX.

List **all** LC-3 instructions that write to the MAR during the evaluate address phase of the instruction cycle, with the Register Transfer description of each.

I’m guessing these are the ones that go through the MARMUX when you walk through the data paths of each of these intructions. All of these are data mov. with the exception of the TRAP which is a data control. Correct me if I am wrong.

LDI	RTN: DR <- Mem[Mem [(PC) + SEXT(IR[8:0])]]
STI	RTN: Mem[Mem [(PC) + SEXT(IR[8:0])]] <- (SR)
LDR	RTN: DR <- Mem[(BaseReg) + SEXT(IR[5:0])]
STR	RTN: Mem[(BaseReg) + SEXT(IR[5:0])] <- (SR)
LD	RTN: DR <- Mem[(PC) + SEXT(IR[8:0])]
ST	RTN: Mem[(PC) + SEXT(IR[8:0])] <- (SR)
TRAP	RTN: R7<-- (PC), MARMUX ← ZEXT(IR[7:0])
JSR	RTN: R7 <- (PC); PC <- SEXT(IR[10:0])

JSRR RTN: R7 <- (PC); PC <- (BaseReg) (where BaseReg is IR[8:6])
do these two go through the MARMUX?
No, because remember JSR/JSRR do not go anywhere near the MARMUX when it gets to the evaluate phase, which is when ADDR1/ADDR2 meet, it goes to the PC instead of the MARMUX.

correct me if i'm wrong but i would like to check up on my add

ADD RTN: DR <-mem[SEXT(IR[8:6] + SEXT(IR[0:2])] ← maybe totally off
or
ADD RTN: DR <- Mem[SEXT(IR[8:6] + SEXT(IR[4:0])] si o no?

DATA PATHS

If any are wrong please correct them!

LD
1. ADDR1: Select PC, selects bit 0
   ADDR2: Selects SEXT(IR[8:0]), selects bit 10
2. MARMUX selects bit 0
3. Gate.MARMUX <- selects 1. i think he wants us to specify when they open especially for gates
4. LD.MDR <- LD.MAR
5. Mem, En/W <-MEM.EN/R
6. LD.MAR <-LD.MDR
(7)*GateMDR
(8)7. LD.REG

LDI
1. ADDR1: Select PC, selects bit 0
   ADDR2: Selects SEXT(IR[8:0]), selects bit 10
2. MARMUX selects bit 0
3. Gate.MARMUX
4. LD.MDR
5. Mem, En/W
6. LD.MAR
7. LD.MDR
8. Mem, En/W
9. LD.MAR
10. LD.REG

LDR

1. SR1MUX: Selects IR[8:6]
2. ADDR1MUX: Selects SR1, bit 1
   ADDR2MUX: Selects SEXT(IR[5:0]), bit 01
3. MARMUX, bit 0
4. GATE.MARMUX
5. LD.MDR
6. MEM En/W
7. LD.MAR
8. LD.REG

STI
1. ADDR1 Selects SEXT(IR[8:0]), bit 10
   ADDR2 Selects PC, bit 0
2. MARMUX, bit 0
3. GATE.MARMUX
4. LD.MDR
5. Do 1-4 again.
6. SR1MUX: Selects IR[11:9]
7. ALU->Pass through ALUK
8. LD.MDR
9. MEM En/W

STR
1. SR1MUX Selects IR[8:6] Base reg
2. ADDR1MUX Selects SR1MUX, bit 1
   ADDR2MUX Selects SEXT(IR[8:6]), bit 10
3. MARMUX, bit 0
4. GATE.MARMUX
5. LD.MDR
6. SR1MUX: Selects [11:9]
7. ALU->ALUK: Pass through
8. Gate.ALU
9. LD.MAR

JSR
1. ADDR1 Select PC, bit 0
2. ADDR2 Select SEXT(IR[10:0]), bit 11
3. PCMUX, select bit 10
4. LD.PC

JSRR
1. GATE.PC   ← someone explain this please. where/why this gate?
2. DRMUX Selects IR[111]
3. SR1MUX Select IR[8:6]
4. ADDR1 Select SR1, bit 1
5. ADDR2 Selects 0, bit 0
6. PCMUX, select bit 10
7. LD.PC

JMP
1. SR1MUX Select IR[8:6]
2. ADDR1MUX Select SR1, bit 1
   ADDR2MUX Selects 0, bit 00
3. PCMUX, bit 01
4. LD.PC

BR
1. ADDR1MUX Select PC, bit 0
   ADDR2MUX Select SEXT(IR[8:0]), bit 10
3. PCMUX, bit 01
4. if(n.N + z.Z + p.P)
5. LD.PC

ADD
1.someone
2.please
3.fill
4.me
5.out
6.for both (2 registers) and (register and hardcode)