Microprocessor Problem Key

M1 - Define: Instruction Set Architecture

M2 - Design the datapath for a computer called the Single-Instruction Computer (SIC) which is able to execute only a single instruction:

sbn a, b, c

which means:

Mem[a] <- Mem[a] - Mem[b] ; if (Mem[a] < 0) PC <- PC + c.

Assume a 16-bit architecture with 48 bit instructions encoded as follows:

[a (16 bits)] [b (16 bits)] [c (16-bit signed offset]

and a memory interface which can only read or write 16 bits at a time.

Clarification: the memory interface looks like the one we used in class and has the same timing

Hint: Each instruction will require six memory acceses: 5 reads and 1 write. You will also find it helpful to have registers in the datapath to store intermediate values while the instruction is executing.

M3 - Design the state machine to control the datapath of the Single-Instruction Computer from the previous problem. You need only show the state diagram.

M1. An ISA is a definition of the instruction set for a processor. It contains explanations of all of the instructions, as well as their bit encodings. It also explains any details of the instructions that depend on the state of the processor.

(Anything like the above will work)

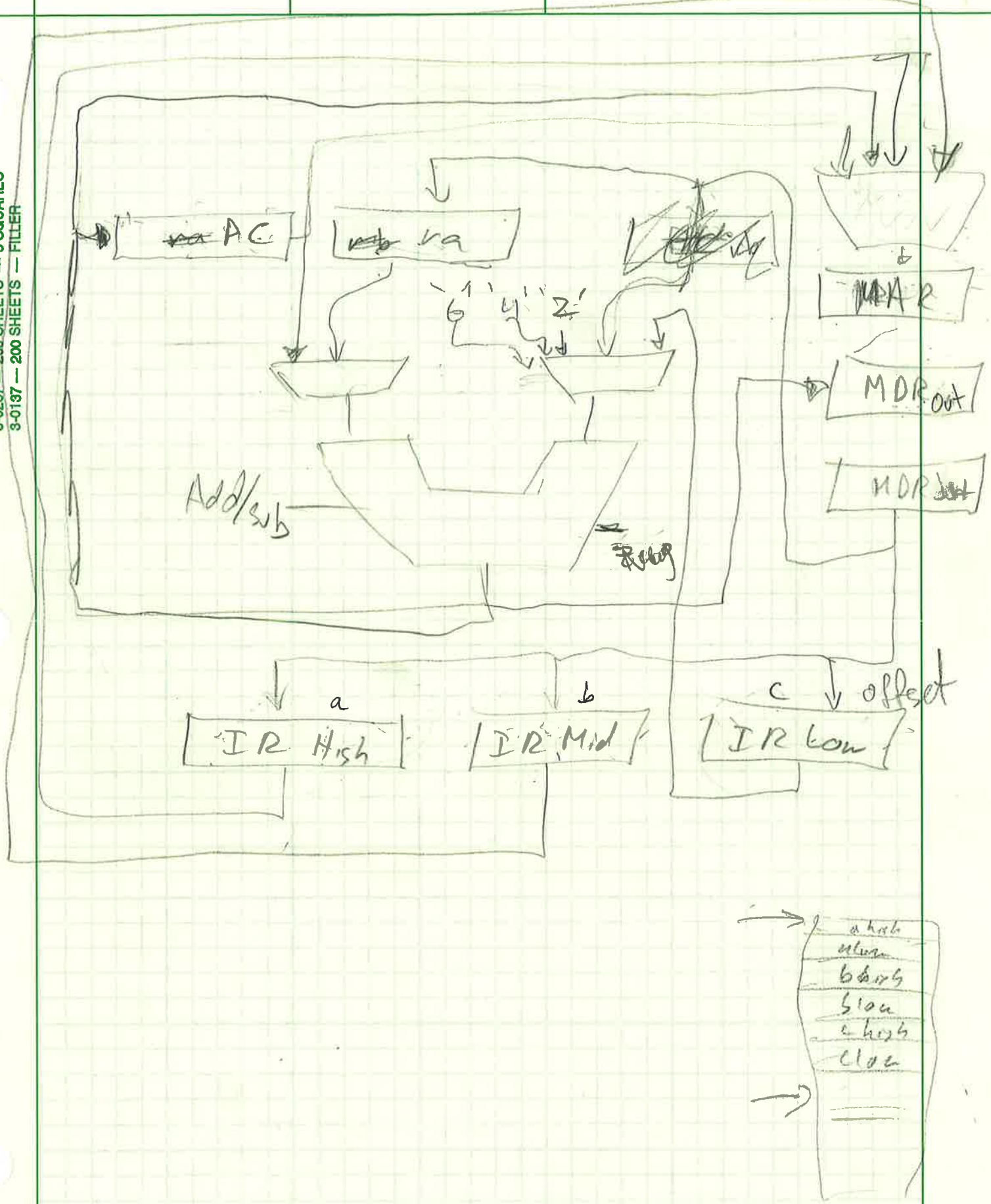
M2. Below are some examples that would work. There are many possible answers. Each of the following is a 3 point section when grading:

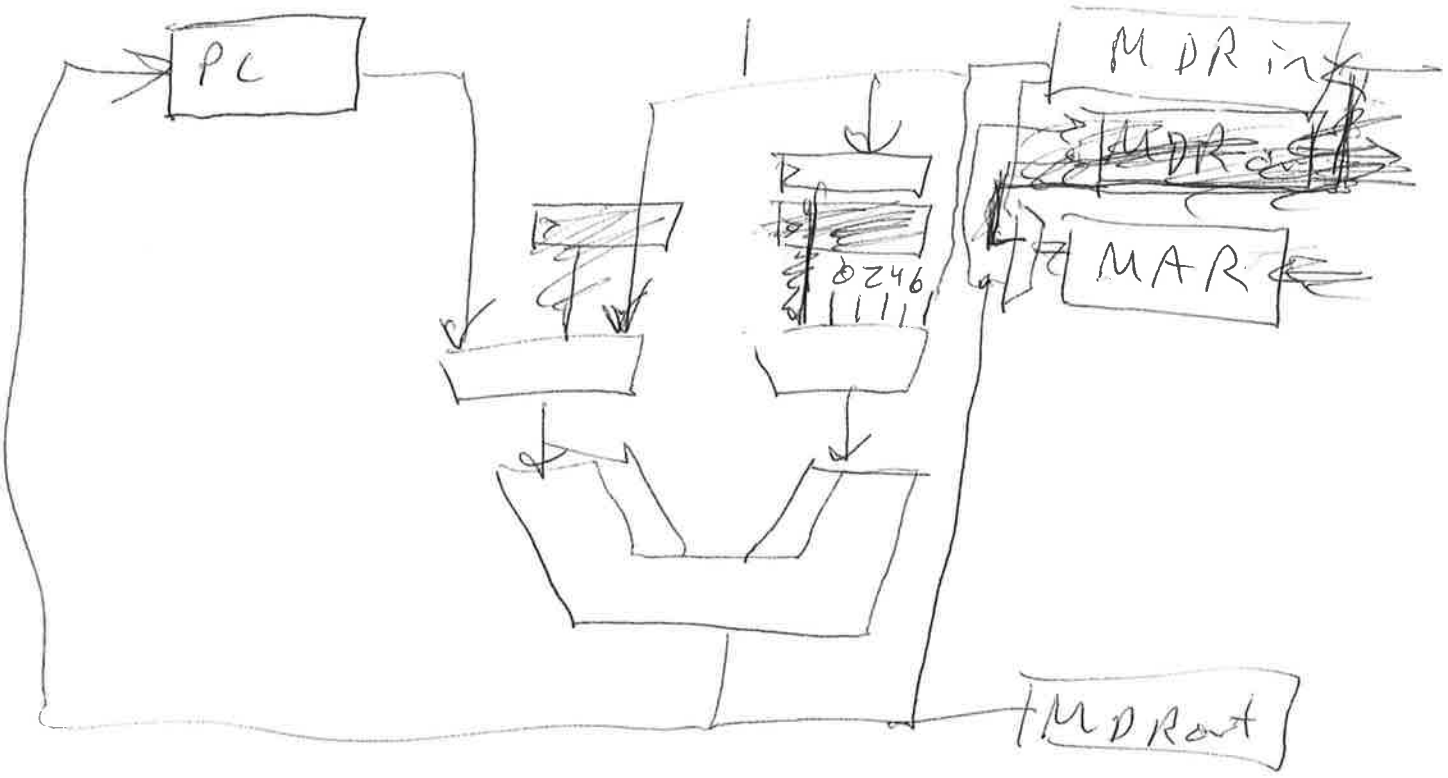
1. Make sure the it is possible to offset the PC correctly to get each part of the instruction and advance to the next one. (Using Inc, Add, and possibly Subtract)
2. Make sure it is possible for them to put things into the MAR
3. Make sure they can get what they need to into the operands
4. Make sure they can write what they need to back into the memory.

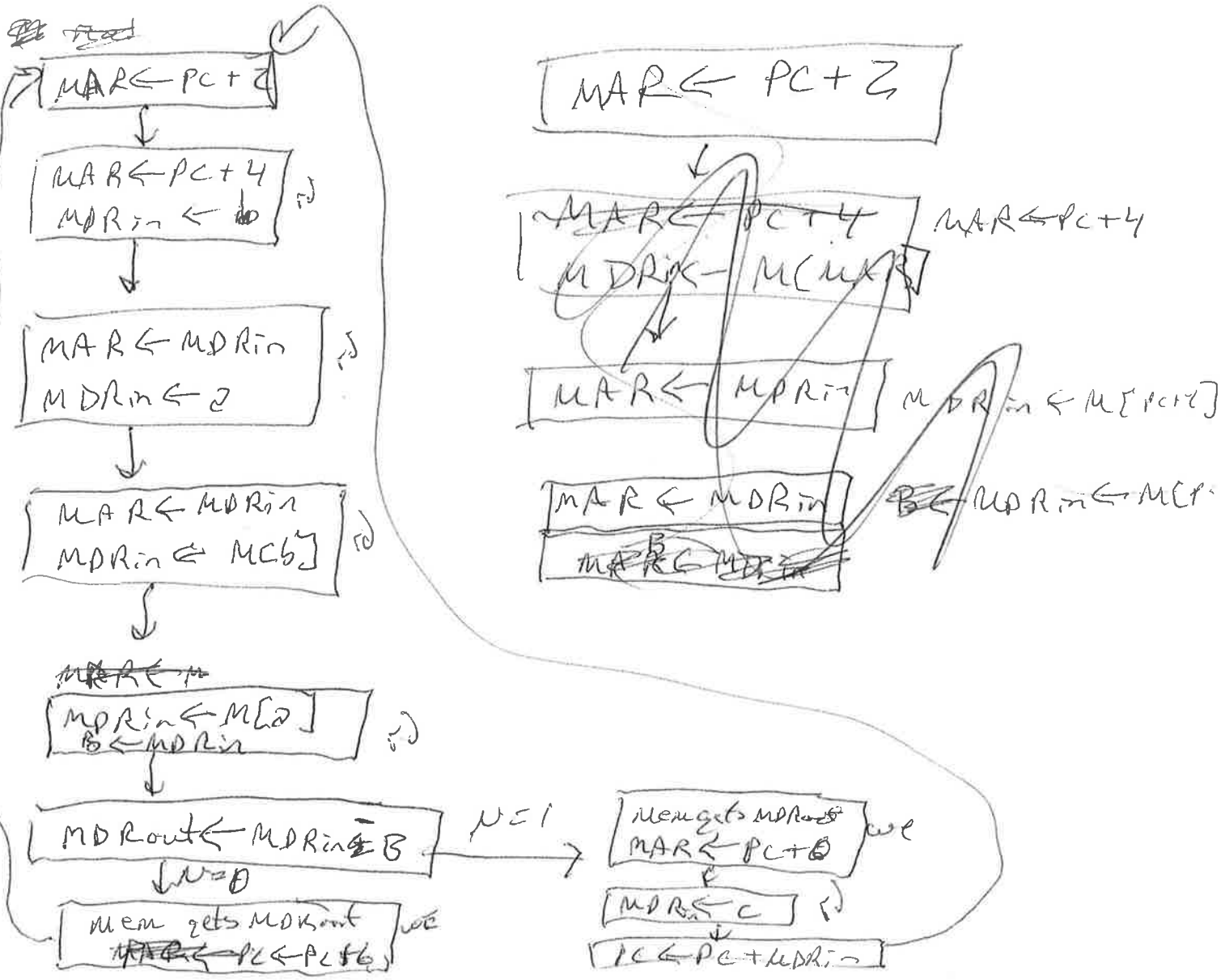
M3. Make sure their state machine will work according to their datapath. The following sections are each worth 3 points:

1-4. Same as 1-4 above but with correct control outputs

5. General State Machine Things: Conflict Free, Complete, Has two possible ways to loop back to fetch corresponding the negative output from Subtract




PC
MDR in
MDR
MAR
0246
MDRout


MAR ← PC + 4
MAR ← PC + 4
MDRin ← M[MAR]
MAR ← MDRin
MDRin ← M[PC+4]
MAR ← MDRin
MDRin ← M[MAR]
MDRin ← M[
MDRin ← M[0]
B ← MDRin
MDRout ← MDRin + B
N = 1
N = 0
Mem gets MDRout
MAR ← PC + B
MDR ← C
PC ← PC + MDRin

1. Fetch - 3 16-bit words
   1 - a PC
   2 - b PC+2
   3 - c PC+4

2. ~~Put~~ a into Mar
   load ~~mar~~ mem into ~~a~~ ~~t~~ ~~1st~~ first operand

3. Put b into mar - MDR
   ~~load~~ put mem @ 2nd operand, @ u

4. Put a addr into mar

5. Store result from subtract to mem

6. If result negative, ~~load~~ add c to PC
   if not add 6. back to fetch.

4 Areas DataPath State Machine.

1. PC
   - Get instructions
   - offset (inc or add)

2. Putting Stuff into the MAR

3. Getting Stuff into the operands

4. Putting data back into the mem

Same areas (4) that include controls for each state

General: completeness
conflict-free
loop back.