1. Adding the REP MOVSB Instruction to the LC-3b [25 points]

State diagram

- \( R = 0 \)
  - SR1 \( \leftarrow \) SR1 - 1
  - [REP]
  - REP = 1
  - MAR \( \leftarrow \) SR2
  - SR2 \( \leftarrow \) SR2 + 1
  - MDR \( \leftarrow \) M[MAR[15:1]'0]
  - R = 1
  - MAR \( \leftarrow \) DR
  - DR \( \leftarrow \) DR + 1
  - \( M[MAR[15:1]'0] \leftarrow \) MDR
  - R = 1
  - To 46

- \( R = 0 \)
  - REP = 0
  - To 18

State Number:
- 10, 46
- 50
- 51
- 40
- 42
- 43
- 44
Modifications to the data path

Additional control signals

- INCDEC/2: PASSR2, +1, -1
- DRMUX/2:
  R7: destination R7
  IR[8:6]: destination IR[8:6]
  IR[2:0]: destination IR[2:0]
- SR1MUX/2:
  IR[8:6]: source IR[8:6]
  IR[2:0]: source IR[2:0]
- COND/3:
  COND_0: Unconditional
  COND_1: Memory Ready
  COND_2: Branch
  COND_3: Addressing Mode
  COND_4: Repeat

2 Pipelining [15 points]

(a) A non-pipelined machine

\[ 9 + 6 + 6 + 9 + 6 + 9 = 45 \text{ cycles} \]

(b) A pipelined machine with scoreboard and five adders and five multipliers without data forwarding
### (c) A pipelined machine with scoreboard and five adders and five multipliers with data forwarding.

| Cycles | 1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28 |
|--------|------------------|
| MUL R3, R1, R2 | F|D|E|E|E|E|W|W |
| ADD R5, R4, R3 | F|-|-|-|-|-|-|-|D |E |E |W |W |
| ADD R6, R4, R1 | F |D |E |E |E |E |W |W |
| MUL R7, R8, R9 | F |D |E |E |E |E |E |W |W |
| ADD R4, R3, R7 | F |-|-|-|-|-|-|-|D |E |E |W |W |
| MUL R10, R5, R6 | F |D |E |E |E |E |E |E |W |W |

**28 cycles** (or 26 cycles with internal register file data forwarding)

### (d) A pipelined machine with scoreboard and one adder and one multiplier without data forwarding

| Cycles | 1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29 |
|--------|------------------|
| MUL R3, R1, R2 | F|D|E|E|E|E|W|W |
| ADD R5, R4, R3 | F|-|-|-|-|-|-|-|D |E |E |W |W |
| ADD R6, R4, R1 | F |-|-|-|-|-|-|-|D |E |E |E |W |W |
| MUL R7, R8, R9 | F |-|-|-|-|-|-|-|D |E |E |E |E |W |W |
| ADD R4, R3, R7 | F |-|-|-|-|-|-|-|D |E |E |E |E |W |W |
| MUL R10, R5, R6 | F |D |E |E |E |E |E |E |E |W |W |

**22 cycles**

### (e) A pipelined machine with scoreboard and one adder and one multiplier with data forwarding

| Cycles | 1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29 |
|--------|------------------|
| MUL R3, R1, R2 | F|D|E|E|E|E|W|W |
| ADD R5, R4, R3 | F|-|-|-|-|-|-|-|D |E |E |W |W |
| ADD R6, R4, R1 | F |-|-|-|-|-|-|-|D |E |E |E |W |W |
| MUL R7, R8, R9 | F |-|-|-|-|-|-|-|D |E |E |E |E |W |W |
| ADD R4, R3, R7 | F |-|-|-|-|-|-|-|D |E |E |E |E |W |W |
| MUL R10, R5, R6 | F |D |E |E |E |E |E |E |E |W |W |

**29 cycles** (or 27 cycles with internal register file data forwarding)

### (e) A pipelined machine with scoreboard and one adder and one multiplier with data forwarding

| Cycles | 1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29 |
|--------|------------------|
| MUL R3, R1, R2 | F|D|E|E|E|E|W|W |
| ADD R5, R4, R3 | F|-|-|-|-|-|-|-|D |E |E |W |W |
| ADD R6, R4, R1 | F |-|-|-|-|-|-|-|D |E |E |E |W |W |
| MUL R7, R8, R9 | F |-|-|-|-|-|-|-|D |E |E |E |E |W |W |
| ADD R4, R3, R7 | F |-|-|-|-|-|-|-|D |E |E |E |E |W |W |
| MUL R10, R5, R6 | F |D |E |E |E |E |E |E |E |W |W |

**23 cycles**
3 Delay Slots [30 points]

(a) What is the number of delay slots needed to ensure correct operation?

2

(b) Which instruction(s) in the assembly sequences below would you place in the delay slot(s), assuming the number of delay slots you answered for part (a)? Clearly rewrite the code with the appropriate instruction(s) in the delay slot(s).

(i) ADD R5 ← R4, R3
    OR R3 ← R1, R2
    SUB R7 ← R5, R6
    J X

    Delay Slots

    LW R10 ← (R7)
    ADD R6 ← R1, R2
    X:

    Solution:

    ADD R5 ← R4, R3
    J X
    OR R3 ← R1, R2
    SUB R7 ← R5, R6
    LW R10 ← (R7)
    ADD R6 ← R1, R2
    X:

(ii) ADD R5 ← R4, R3
    OR R3 ← R1, R2
    SUB R7 ← R5, R6
    BEQ R5 ← R7, X

    Delay Slots

    LW R10 ← (R7)
    ADD R6 ← R1, R2
    X:

    Solution:

    ADD R5 ← R4, R3
    SUB R7 ← R5, R6
    BEQ R5 ← R7, X
    OR R3 ← R1, R2
    NOP
    LW R10 ← (R7)
    ADD R6 ← R1, R2
    X:

(iii) ADD R2 ← R4, R3
    OR R5 ← R1, R2
    SUB R7 ← R5, R6
    BEQ R5 ← R7, X

    Delay Slots

    LW R10 ← (R7)
    ADD R6 ← R1, R2
    X:
Solution:

ADD R2 <- R4, R3
OR R5 <- R1, R2
SUB R7 <- R5, R6
BEQ R5 <- R7, X
NOP
NOP
LW R10 <- (R7)
ADD R6 <- R1, R2
X:

(c) Can you modify the pipeline to reduce the number of delay slots (without introducing branch prediction)? Clearly state your solution and explain why.

Move the resolution of jump targets and branch targets and destinations to the decode stage.
Jumps and branches would get resolved one cycle earlier and hence one delay slot would be enough to ensure correct operation.

4 Hardware vs. Software Interlocking [30 points]

(a) Calculate the number of cycles it takes to execute each of these two programs on Machines X and Y.

Program A:

Machine X:
The execution timeline for this reordered code is as below:

```
F D E E E W W
F D E E E W W
F D E E E W W
F D E E E W W
F D E E E W W
```

23 cycles

Machine Y:
The compiler places nops between every two instructions if they are dependent and it can’t find other independent instructions.

```
ADD R5 <- R6, R7
NOP
NOP
NOP
ADD R3 <- R5, R4
NOP
NOP
NOP
ADD R6 <- R3, R8
NOP
NOP
NOP
ADD R12 <- R3, R7  // instruction reordered
ADD R9 <- R6, R3
```
The execution time line of this modified code segment is as shown below. N indicates a NOP in a pipeline stage.

```
F D E E E W W
N N N N N N N
N N N N N N N
N N N N N N N
N N N N N N N
F D E E E W W
N N N N N N N
N N N N N N N
N N N N N N N
N N N N N N N
F D E E E W W
N N N N N N N
N N N N N N N
N N N N N N N
N N N N N N N
F D E E E W W
F D E E E W W
```

22 cycles

Program B:

Machine X:
The execution timeline is as below:

```
F D E E E W W
F D E E E W W
F D E E E W W
F D E E E W W
F D E E E W W
```

16 cycles

Machine Y:
The compiler reorders instructions such that each pair of dependent instructions is separated by four independent instructions, or nops if not enough independent instructions can be found.

```
ADD R4 <- R5, R6
ADD R8 <- R9, R10
ADD R3 <- R1, R2
NOP
NOP
ADD R7 <- R1, R4
ADD R12 <- R8, R2
ADD R14 <- R8, R4
```

The execution timeline for this reordered code is as below:
(b) Which machine takes a smaller number of cycles to execute each Program A and Program B?

**Solution:**
Machine Y takes a smaller number of cycles for both Programs.

(c) Does the machine that takes the smaller number of cycles for Program A also take the smaller number of cycles than the other machine for Program B? Why or why not?

**Solution:**
Yes, Machine Y takes a smaller number of cycles as the compiler can find independent instructions to place between dependent instructions.

(d) Would you say that the machine that provides a smaller number of cycles as compared to the other machine has higher performance (taking into account all components of the performance equation we discussed in class)?

**Solution:**
For both programs, machine Y has higher performance than machine X, as machine Y takes a smaller number of cycles to execute and also has simpler hardware than machine X (no need to perform interlocking in hardware).

(e) Calculate the instruction code size of each of these two code segments on Machine X and Y, assuming a fixed-length ISA, where each instruction is encoded as 4 bytes.

**Solution:**
Program A:
- Machine X - 20 bytes
- Machine Y - 64 bytes (because of the additional NOPs)

Program B:
- Machine X - 24 bytes
- Machine Y - 32 bytes (because of the additional NOPs)

(f) Which machine incurs lower instruction code size for Program A and Program B?

**Solution:**
For both programs, machine X has lower instruction code size than machine Y.

(g) Does the same machine incur lower instruction code sizes for both Program A and Program B? Why or why not?
Solution:
Yes.
For both programs, machine X has lower code size as the compiler inserts NOPs in machine Y.