Answer 4

A1

1. `0101 011 010 1 00100` ，如果结果是 0，则 2 号机器处于忙碌状态
2. `0101 011 010 1 01100` ，如果结果为 0，则 2 号和 3 号机器均处于忙碌状态
3. `0101 011 010 1 11111` 或者 `0101 011 010 0 00 010`，如果结果为 0，则全部 8 台机器均处于忙碌状态，此处要求 `R2` 的前 8 位均为 0
4. 不能使用一条指令，因为 0101 的立即数最多只有五位，但可以考虑先将需要的掩码 `0000 0000 0100 0000` 存入寄存器，再使用 `0101`（`AND`）的寄存器与模式，则可以检测6号机器的工作状态

A2

如下，先将值 x3101 存到 `R1` 中，再把 M[`R1` +2] 存到 `R2` 中，也就是 x1482.

Address	Value	Comments
x30FF	1110 001 000000001	LEA R1, #1
x3100	0110 010 001 000010	LDR R2, R1, #2
x3101	1111 0000 0010 0101	TRAP x25
x3102	0001 0100 0100 0001	x1441
x3103	0001 0100 1000 0010	x1482

A3

1. 如下两条即可.

```
LDR R2, R1, #0
STR R2, R0, #0
```

2. 如下微指令会在 `MOVE` 被解码后执行.

```
MAR <- SR
MDR <- Memory[MAR]
MAR <- DR
Memory[MAR] <- MDR
```

> 注：题意写得不清晰，如果有直接 `STR R1, R0, #0` 的也可以算对，此时第2小问应为 `MDR <- SR; MAR <- DR; Memory[MAR] <- MDR`

A4

Address	Instruction	Comments
x3000	1001 100 001 111111	NOT R4, R1
x3001	**0101 100 100 000 010**	AND R4, R4, R2
x3002	1001 011 010 111111	NOT R3, R2
x3003	**0101 011 011 000 001**	AND R3, R3, R1
x3004	1101 011 011 000 100	OR R3, R4, R3

A5

Addressing modes: register, immediate, PC-relative, indirect, Base+offset

Instruction	Type	Addr mode(s)
ADD	operate	register, immediate
NOT	operate	register
LEA	data movement	immediate
LDR	data movement	Base+offset
JMP	control	register

A6

1. `ADD R4, R5, #0` or `AND R4, R5, #-1`
2. `AND R3, R3, #0`

3. `NOT R1, R7`, `ADD R1, R1, #1`, `ADD R1, R1, R6`
4. `LD R1, DATA`, `ADD R1, R1, R1`, `ST R1, DATA`
5. `ADD R1, R1, #0` or `AND R1, R1, #-1`

A7

- 对于 `JMP` 指令，LC-3 需要 1 次内存访问。由于其内部内部存储的为下一条指令相对于当前地址的偏移量，除了本身的取指之外无须再进行内存访问。
- 对于 `ADD` 指令，LC-3 需要 1 次内存访问。其第一个操作数必须从寄存器中获得，第二个操作数可以为立即数或者使用寄存器寻址，故除了本身的取指外无须再进行内存访问。
- 对于 `LDI` 指令，LC-3 需要 3 次内存访问。第一次是本身的取指，第二次是获取内存中 incremented PC + offset 位置的值，第三次是用那个值再去内存中取值。

A8

1. 先使用 `LEA` 指令，偏移量为 x3E，计算得到地址 0x304F 放入 `R3`.

再使用 `LDR` 指令，偏移量为 x01，从 `R3` 中读取地址计算，得到地址 0x3050，读取内存将 0x70A2 放入 `R4`.

再使用 `LDR` 指令，偏移量为 x01，从 `R4` 中读取地址计算，得到地址 0x70A3，读取内存将 0x70A3 放入 `R7`.

再使用 `LDR` 指令，偏移量为 11 1111，从R7中读取地址计算，得到地址 0x70A2，读取内存将 0x70A4 放入 `R6`.

2. 若使用 `LEA`，由于偏移量位数不足，无法计算得到目标值.

A9

指令序列如下

Address	Value	Comments
x3000	0101 0000 0010 0000	AND R0, R0, #0
x3001	0101 1111 1110 0000	AND R7, R7, #0
x3002	0001 1100 0010 0001	ADD R6, R0, #1
x3003	0001 1101 1000 0110	ADD R6, R6, R6
x3004	0101 1001 0100 0110	AND R4, R5, R6
x3005	0000 0100 0000 0001	BRz x3007
x3006	0001 0000 0010 0001	ADD R0, R0, #3
x3007	0001 1111 1110 0010	ADD R7, R7, #2
x3008	0001 0011 1111 0010	ADD R1, R7, #-14
x3009	0000 1001 1111 1001	BRn x3003
x300A	0101 1111 1110 0000	AND R7, R7, #0

```
r0 = r0 & 0;
r7 = r7 & 0;
r6 = r0 + 1;
do {
  r6 = r6 + r6;
  r4 = r5 & r6;
  if (r4 != 0) {
    r0 += 3;
  }
  r7 = r7 + 2;
  r1 = r7 - 14;
} while(r1 < 0);
r7 = r7 & 0;
```

反推得到 `R5` 的第 2 位到第 8 位 (最低位为第 1 位) 中有且仅有 4 位为 1.

A10

依次分析每行命令。

- x3000, x3001 将 `R0` 中的值取负数存在 `R0` 中，这用来判断 `R1` 和 `R0` 是否相同，也就是用 `R1` 加上 `-R0` 是否为 0 来判断

- x3002 将 `R2` 中的值置为 0
- x3003 将 `R3` 中的值置为 `R0` 和 `R1` 的和，也就是 `R1` 减去原来的 `R0`，那么接下来的 x3004 应该是一个判断是否为 0 跳转的命令，跳转目标现在还不知道
- x3005 是 `R2` 加 1，之后 x3006 未知，而 x3007 判断若 x3006 结果为 0 则跳转到 x300F. x300F, x3010 将 `R2` 清空后减 1，可以看出这是判断已经不可能让原本的 `R1` 左移成 `R0` 后将结果 -1 存在 `R2` 里，则 x3011 将 `R2` 中值存到 x3020 中，即 ST R2, x3020，这时也可以判断之前的 x3004 应跳转到这里，也就是 BRz x3011
- x3008, x3009 若 `R1` 为负数则跳转到 x300C, x300D 然后跳转到 x3003
- 若 `R1` 为正数，x300A 将 `R1` 翻倍后跳转到 x3003，否则经 x300C, x300D 后跳转到 x3003，而此时为 `R1` 为负数情况，也就是最高位为 1，所以将 `R1` 翻倍后要在最低位加 1，这样就实现了左移 1 位的操作
- 最后思考剩下的 x3006，因为左移 16 位就回到本身，所以如果 `R2` 加到 16 就说明已经不可能左移成 `R0` 了，所以 x3006 应该是判断 `R2` 是否为 16，也就是 `ADD R3, R2, #-16`

Address	Value	Comments
x3000	1001 000 000 111111	NOT R0, R0
x3001	0001 000 000 1 00001	ADD R0, R0, #1
x3002	0101 010 010 1 00000	AND R2, R2, #0
x3003	0001 011 000 0 00 001	ADD R3, R0, R1
x3004	**0000 010 000001100**	BRz x3011
x3005	0001 010 010 1 00001	ADD R2, R2, #1
x3006	**0001 011 010 1 10000**	ADD R3, R2, #-16
x3007	0000 010 000000111	BRz x300F
x3008	0101 001 001 1 11111	AND R1, R1, xFFFF
x3009	0000 100 000000010	BRn x300C
x300A	0001 001 001 0 00 001	ADD R1, R1, R1
x300B	0000 111 111110111	BRnzp x3003
x300C	**0001 001 001 0 00 001**	ADD R1, R1, R1
x300D	**0001 001 001 1 00001**	ADD R1, R1, #1
x300E	0000 111 111110100	BRnzp x3003
x300F	0101 010 010 1 00000	AND R2, R2, #0
x3010	0001 010 010 1 11111	ADD R2, R2, #-1
x3011	**0011 010 000001110**	ST R2, x3020
x3012	1111 0000 0010 0101	TRAP x025