1(a)

W	X	Y	Z	b
T	T	T	T	1
T	T	T	F	1
T	T	F	T	1
T	T	F	F	1
T	F	T	T	1
T	F	T	F	0
T	F	F	T	0
T	F	F	F	1
F	T	T	T	1
F	T	T	F	1
F	T	F	T	1
F	T	F	F	0
F	F	T	T	1
F	F	T	F	0
F	F	F	T	0
F	F	F	F	0

0 1 2 3 4 5 6 7 8 9 A B C D E F

1.(b)

WX \ YZ	00	01	11	10
00	0	0	0	1
01	1	1	1	1
11	1	1	1	1
10	1	0	1	0

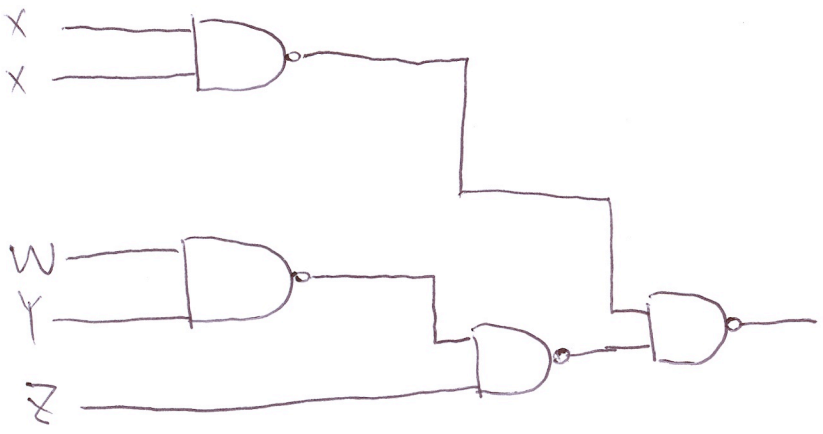
1.(c)

$x + w'yz' + wy'z' + wyz$.

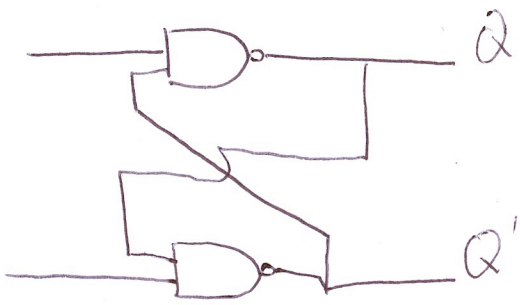
~~$= (W \text{ NAND } Y \text{ NAND } Z) \text{ NAND } (X \text{ NAND } X)$~~

$= (W \text{ NAND } Y \text{ NAND } Z) \text{ NAND } (X \text{ NAND } X)$

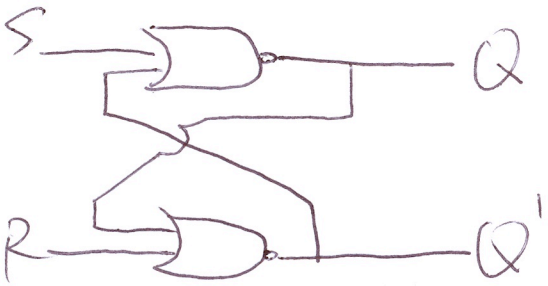
1.(d)



a)



b)



3.

```
int a = 5          // a is storted in data
int b[20]          // b is bss
int main( )        // stored in text
{  int x           // x stored stack
   int *p=(int*)malloc(sizeof(int));
                   // stored in heap
```

① The loader is a program that is used to run an executable file in a process.

② Before the program starts running, the loader allocates space for all the sections of the executable file (text, data, bss etc)

③ It loads into memory the executable and shared libraries

④ It also writes (resolves) any values in the executable to point to the functions/variables in the shared libraries

⑤ Once memory image is ready, the loader jumps to the _start entry point that calls init() of all libraries and initializes static constructors. Then it calls main() and the program begins

⑥ _start also calls exit() when main() returns.

5. "Dynamic Linker" is the part that loads and links the shared libraries for an executable when it is executed. Linking is often referred to as a process that is performed at compile time of the executable while a dynamic linker is in actuality a special loader that loads external shared libraries into a running process and then binds those shared libraries dynamically to the running process.

"static linker" copies all Librarry routines used in the program into executable file. It need more space and memory but it is faster.

6. 1001010 x 1101

```
  1001010
     1101
  -------
  1001010
 1001010
1001010
-------
1111000010
```

$2^1 + 2^6 + 2^7 + 2^8 + 2^9 = 962$

7. 1011) 100101001

```
          11011
     -----------
1011 ) 100101001
       1011
       ----
        1111
        1011
        ----
       10000
        1011
        ----
        1011
        1011
        ----
           0
```

11011 = 27

8. Val in binary 0011 1111 1110 1000 0000 0000 0000 0000

9. Pipe line allows the execution of instruction in parallel
Pipe stall occur if the next instruction depends on the result of the previous instruction.
Avoid: A programmer can delay the use of results by
① reordering the instructions
② Avoid introducing unnecessary branches.
③ Delay references to result register.

10.

-58

$\therefore$ 58 : 00111010

-58 : 11000110

78 : 01001110

78	01001110
+(-58)	11000110
	00010100

$= 20.$

11.
```
int isBigEndian(){
    int k = 8;
    char *p = (char*)&k;
    if(p*==8) return 0;
    return 1;
}.
```

12. Harvard Architecture: has physically separate signals and storage for code and data. It is possible to access program memory and data memory simultaneously.

Von Newman Architecture: have shared signals and memory for code and data. Thus, the program can be easily modified by itself since it is stored in read-~~wr~~ write memory.

CISC stand for complex instruction set computer.
In common, CISC chips are relatively slow (compared to RISC chips) per instruction, but use little (less than RISC) instructions. RISC is cheaper and faster ~~and~~. RISC puts a greater burden on the software.

14. Program Memory (kB) : 32.
RAM (KB) : 2
Clock speed : 20M
~~Number~~ Number of I/O ports : 23
Number of A/D ports: 6.

```
lds   r20, b
lds   r21, (b)+1
lds   r18, a
lds   r19, (a)+1
add   r20, r18
abc   r21, r19
sts   (c)+1  r25
sts   c   r24
ret
```

```
lds   r20, b
lds   r21, (b)+1
lds   r18, a
lds   r19, (a)+1
mul   r20, r18
movw  r24, r0
mul   r20  r19
add   r25, r0
mul   r21, r18
add   r25, r0
clr   r1
sts   (c)+1, r25
sts   c, r24
ref.
```

16.

```
    .text
.globl  addarray
        .type   addarray, @function.
addarry :
        movl    (%esi), %eax
        movl    %edi,   %edx
while : ~~comp~~
        cmpl    $0 ,    %edx
                jne     loop
                ret
loop:   ~~addl~~
        subl    $1 ,    %edx
        add     $4 ,    %~~ed~~esi
        add     (%esi), %eax
        jmp     while.
```

17

```
str1:
        .string "%d"
str2:
        .string "%d\n"
print1:
        .string "a=%d\n"
print2:
        .string "b=%d\n"
print3:
        .string "max=%d\n"
print4:
        .string "min=%d\n"
print5:
        .string "avg=%d\n"

.text
.globl main
main:
        pushq   %rbp
        movq    %rsp,   %rbp
        subq    $80,    %rsp
        movq    $0,     -8(%rbp)

        movq    -8(%rbp),       %rax
        leaq    -32(%rbp,%rax,8),       %rsi
        movq    $str1,  %rdi
        call    scanf

        addq    $1,     -8(%rbp)

        movq    -8(%rbp),       %rax
        leaq    -32(%rbp,%rax,8),       %rsi
        movq    $str1,  %rdi
        call    scanf

        movq    -32(%rbp),      %rsi
        movq    $print1,        %rdi
        movq    $0,     %rax
        call    printf

        movq    -24(%rbp),      %rsi
        movq    $print2,        %rdi
        movq    $0,     %rax
        call    printf

        movq    -32(%rbp),      %rdx
        movq    -24(%rbp),      %rcx

        cmpq    %rdx,   %rcx
        jge     min
        jmp     max


max:
        movq    -32(%rbp),      %rsi
        movq    $print3,        %rdi
        movq    $0,     %rax
        call    printf

        movq    -24(%rbp),      %rsi
        movq    $print4,        %rdi
        movq    $0,     %rax
        call    printf

        movq    -24(%rbp),      %rax
        movq    -32(%rbp),      %rdx
        addq    %rdx,   %rax
        shrq    $1,     %rax
```

```
        movq    $print5,        %rdi
        movq    %rax,   %rsi
        movq    $0,     %rax
        call    printf

        leave
        ret

min:
        movq    -24(%rbp),      %rsi
        movq    $print3,        %rdi
        movq    $0,     %rax
        call    printf

        movq    -32(%rbp),      %rsi
        movq    $print4,        %rdi
        movq    $0,     %rax
        call    printf

        movq    -24(%rbp),      %rax
        movq    -32(%rbp),      %rdx
        addq    %rdx,   %rax
        shrq    $1,     %rax


        movq    $print5,        %rdi
        movq    %rax,   %rsi
        movq    $0,     %rax
        call    printf

        leave
        ret
```