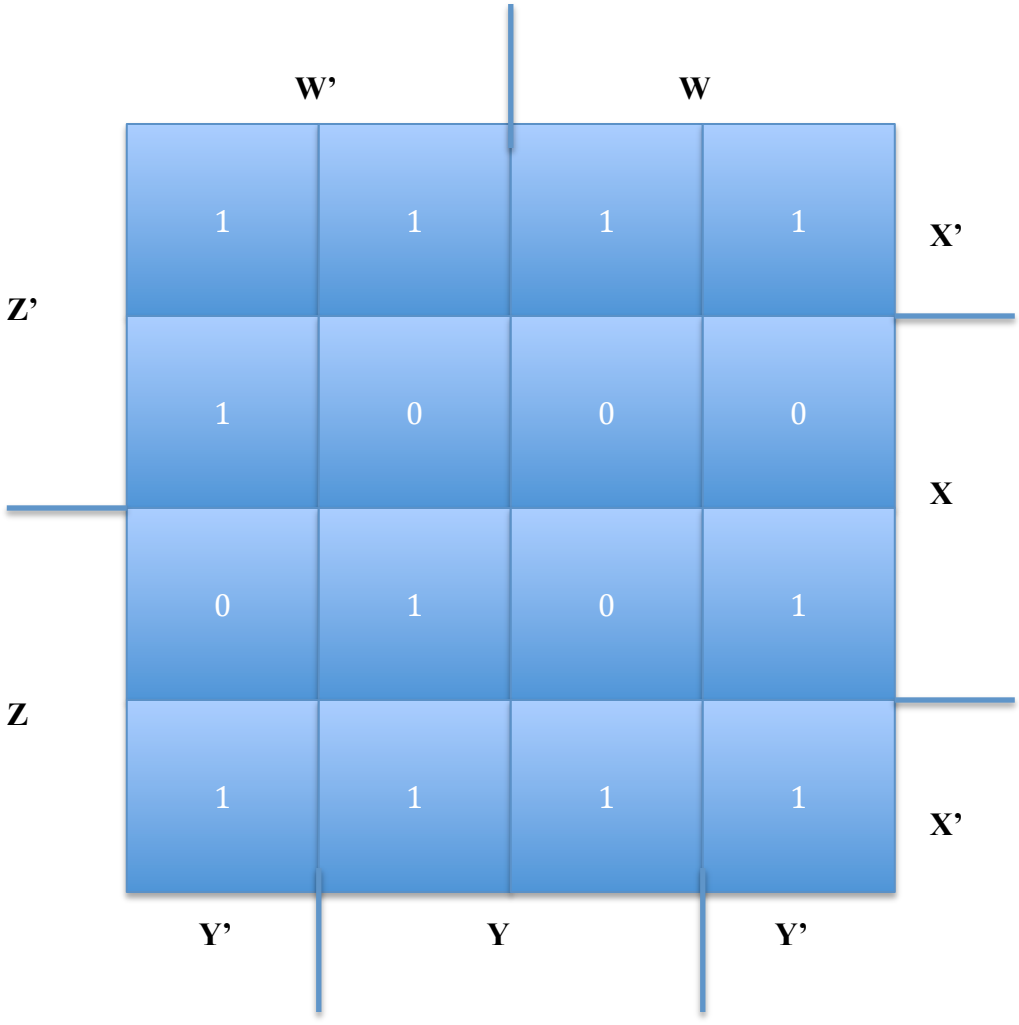
Kritin Gokharu
kgokharu@purdue.edu

CS 250: Homework 1

1. a.

W	X	Y	Z	b
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

b.



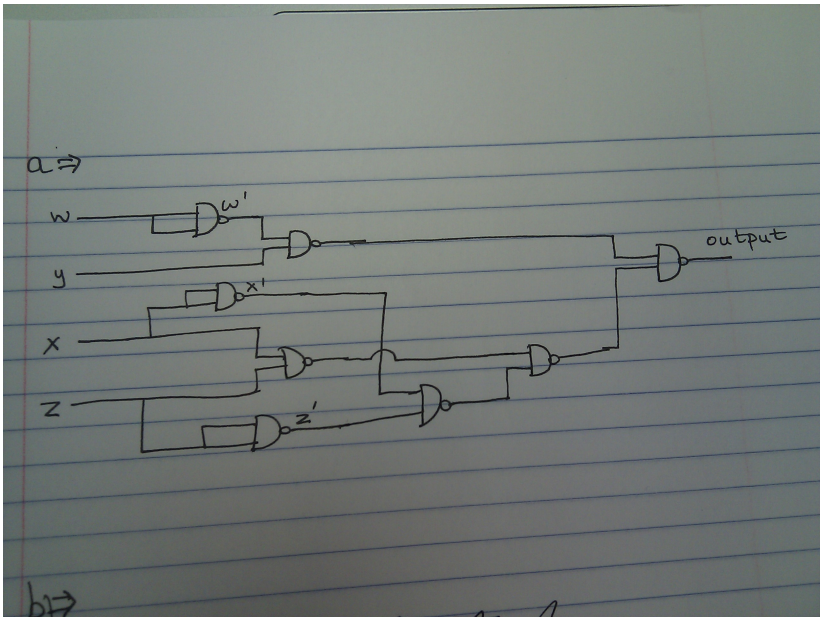
Kritin Gokharu
kgokharu@purdue.edu

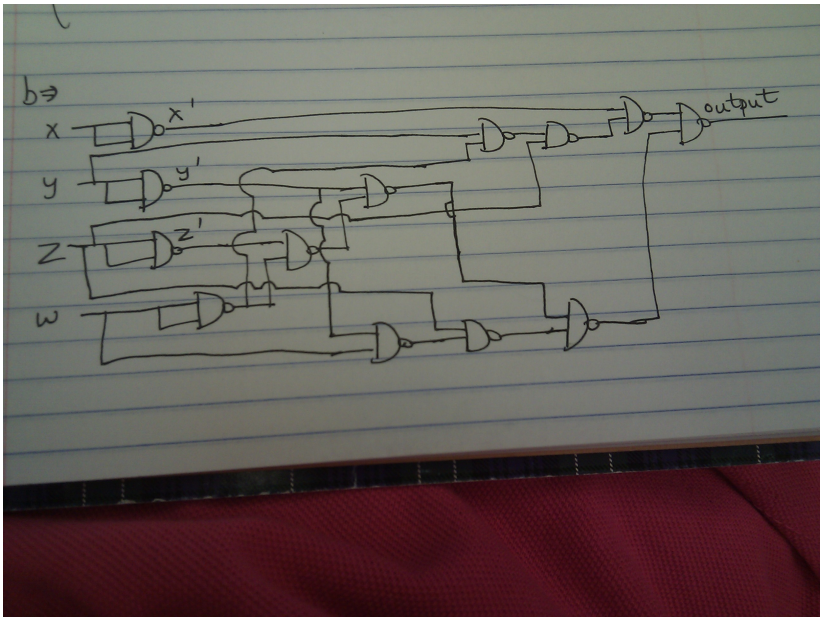
c.

$a = w + (w'y) + (x'z') + (xz)$

$b = x' + (w'y'z') + (w'yz) + (wy'z)$

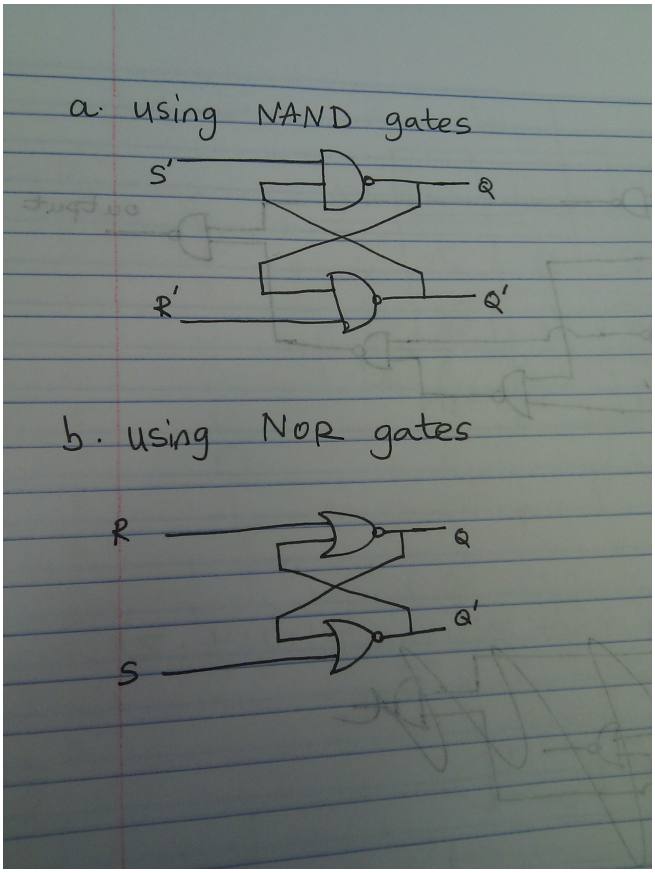
d.





Kritin Gokharu
kgokharu@purdue.edu

2.



3.

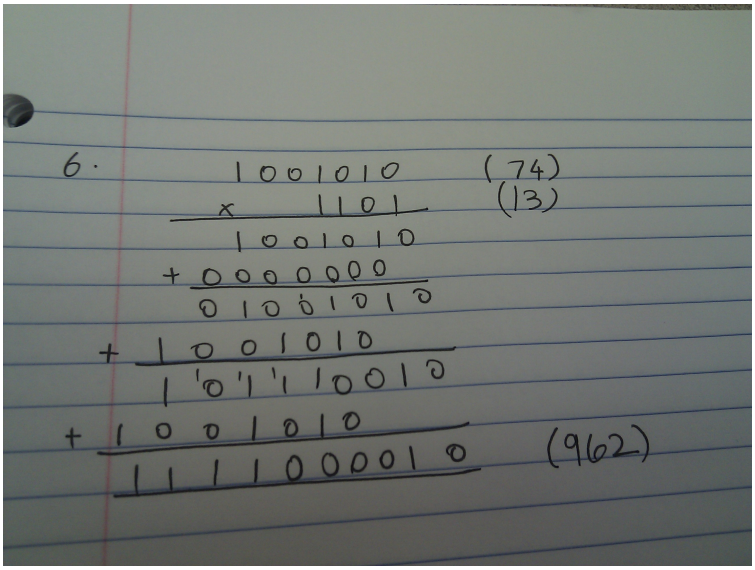
```
int a = 5;  // a is stored in data

int b[20];  // b is stored in bss

int main() {  // main is stored in text
{
   int x;    // x is stored in stack

   int *p = (int *)malloc(sizeof(int)); // p points to memory stored in heap
}
```

4. Before the program starts running, the loader allocates space for all the sections of the executable file (text, data, bss, etc). It loads the executable and shared libraries into memory. Once the memory image is ready, the loader jumps to the _start entry that calls the init() function of all libraries and initializes static constructors. Then it calls main and the program begins. _start also calls exit() when main() returns.

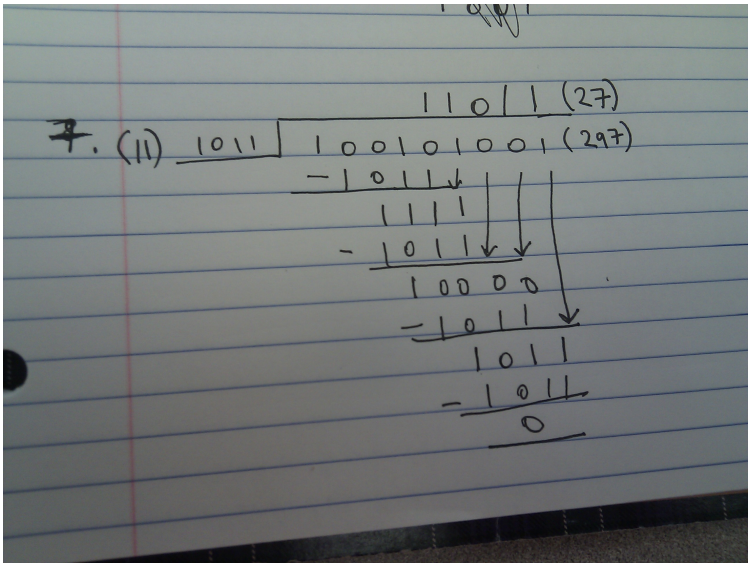
Kritin Gokharu
kgokharu@purdue.edu

5. Static linker copies the shared libraries into the executable image of the file, whereas the dynamic linker copies the name of the shared library into the executable and links the routines only during runtime.

6.



7.



8.
s = 0
e = 01111111111
m = 010000000…

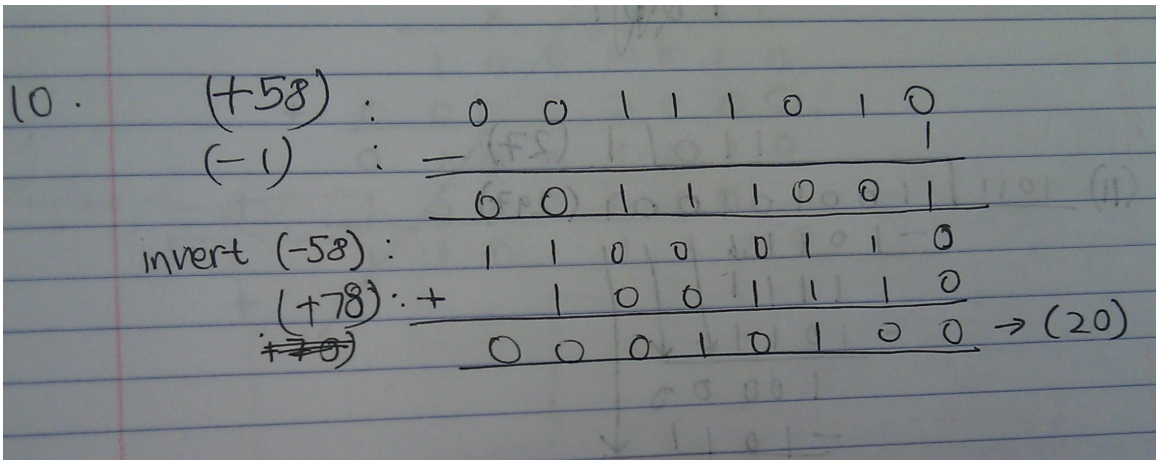
Therefore, the binary representation is

00111111111101000000000…

Kritin Gokharu
kgokharu@purdue.edu

9.
A pipe stall occurs when a process cannot be started because it requires input from a preceding process that has yet to be completed. It can be avoided by reordering instructions to ensure that instructions, which use registers that require input from other registers, are delayed.

10.



11.

```
int isBigEndian() {
        int i = 5;
        char * p  = (char*) &i;
        if(*p!=5)
                return 1;
        else
                return 0;
}
```

12.
Under Harvard architecture, the CPU can both read an instruction and perform a data memory access at the same time, whereas under Von Newman architecture, the CPU cannot read an instruction and perform transfer of data at the same time.

Kritin Gokharu
kgokharu@purdue.edu

13.
CISC – Complex Instruction Set Computer
- has many complex instructions
- instructions are a combination of simple instructions
- instructions take different times to complete

RISC – Reduced Instruction Set Computer
- has simpler instructions
- Instructions have same length, so take same amount of time to complete

14.

ATMEGA328
Program Memory(KB): 32
RAM (KB): 2
Clock Speed (Hz): 16000000
Number of I/O ports: 12
Number of A/D ports: 6

14.

```
.data
        movq $6, %rdi
        movq $3, %rdx
        movq $0, %rax
.globl main
main:
        addq %rdi, %rdx
        movq %rdx, %rax
        ret
```

15.

```
.data
        movq $6, %rdi
        movq $4, %rdx
        movq $0, %rax
.globl main
main:
        imulq %rdi, %rdx
        movq %rdx, %rax
        ret
```

Kritin Gokharu
kgokharu@purdue.edu

16.

```
.text
.globl addarray
        .type addarray, @function
addarray:
        movl $0, %eax
        movq $0, %rdx
L1:     cmpq %rdx, %rdi
        je L2
        addq $1, %rdx
        add (%rsi), %rax
        add $4, %rsi
        jmp L1
L2:     ret
```

17.

```
.data
        movq $2, %r8
        movq $0, %rbx
        movq $0, %rdx
        formatString: .string “%d”
        formatStringP: .string “%d\n”
.text
        .globl main
main:
        movq $formatString, %rdi
        movq %rbx, %rsi
        movq $0, %rax
        call scanf
        movq $formatString, %rdi
        movq %rdx, %rsi
        movq $0, %rax
        call scanf

        cmpq %rbx, %rdx
        jg disp1
        cmpq %rbx, %rdx
        jg disp2
disp1:  movq $formatStringP, %rdi
        movq (%rbx), %rsi
        movq $0,%rax
        call printf
        movq $formatStringP, %rdi
```

```
        movq (%rdx), %rsi
        movq $0,%rax
        call printf
        jmp final
disp2: movq $formatStringP, %rdi
        movq (%rdx), %rsi
        movq $0,%rax
        call printf
        movq $formatStringP, %rdi
        movq (%rbx), %rsi
        movq $0,%rax
        call printf
        jmp final
final:  addq %rdx,%rbx
        idivq %rbx, %r8
        movq %r8, %rax
        ret
```