

ECEN 449: Microprocessor System Design  
Department of Electrical and Computer Engineering  
Texas A&M University

Assignment #2  
Solutions

1. Write a program to multiply two matrices  $A$  and  $B$  of unknown sizes. The sizes of the matrices will be given as input. The input matrices will be given in two files inputA.txt and inputB.txt. The first line of the input file will contain two integers: number of rows, number of columns of the matrix in the file. The entries of the matrix will be listed on the next line in a row-major order. Assume the entries of the matrix are floating point numbers. Print the product matrix  $C$  into an output file outputC.txt in the same format as the input files. Provide comments in your code.

```
Solution. #include <stdio.h>
int main()
{
FILE *fileA, *fileB, *fileC; //file pointers
int rowsA = 0, colsA = 0, rowsB = 0, colsB = 0; //dimension vars
int i, j, k; //iterators
float **inpA, **inpB, **outC;
float sum;
float temp;
//check if files can be opened
if((fileA = fopen("inputA.txt", "r")) == NULL){
printf("Cannot Open File\n"); //error message
exit(1); //exit program
}
if((fileB = fopen("inputB.txt", "r")) == NULL){
printf("Cannot Open File\n"); //error message
exit(1); //exit program
}
}
```

```

//read file
fscanf(fileA,"%d %d" ,&rowsA, &colsA); //get dimensions of A
//Dynamically Allocate memory to input matix-A
inpA = malloc(rowsA*sizeof(float *));
for(i=0;i<rowsA;i++)
inpA[i] = malloc(colsA*sizeof(float));
//for each entry, read data from file
for(i = 0; i < rowsA; i++){ //iterate through rows
for(j = 0; j < colsA; j++){ //iterate through cols
if(fscanf(fileA, "%f",&temp)!=1) //make sure it is there
break;
inpA[i][j] = temp;
}
}
//read file inputB
fscanf(fileB,"%d %d" ,&rowsB, &colsB); //get dimensions of B
//Dynamically Allocate memory to input matix-B
inpB = malloc(rowsB*sizeof(float *));
for(i=0;i<rowsB;i++)
inpB[i] = malloc(colsB*sizeof(float));
for(i = 0; i < rowsB; i++){ //iterate through rows
for(j = 0; j < colsB; j++){ //iterate through cols
if(fscanf(fileB, "%f",&temp)!=1) //make sure it is there
break;
inpB[i][j] = temp;
}
}
//print A to screen=====
printf("\nMatrix A: %d x %d\n", rowsA, colsA);
for(i = 0; i < rowsA; i++){
for(j=0; j<colsA;j++){
printf("%f ", inpA[i][j])
}
printf(" \n");
}

//print B to screen=====
printf("\nMatrix B: %d x %d\n", rowsB, colsB);
for(i = 0; i < rowsB; i++){

```

```
for(j=0; j<colsB;j++){
printf("%f ", inpB[i][j]);
}
printf("\n");
}
printf("\n");
//DO MULTIPLICATION!!!!=====
//Check if the dimensions are correct to perform matrix multiplication i.e. colA = rowB
if(colsA != rowsB){
printf("Can't Do multiplication, rows of A and cols of B are !=\n");
exit(1);
}
//Dynamically Allocate memory to result matrix
outC = malloc(rowsA*sizeof(float *));
for(i=0;i<rowsA;i++)
outC[i] = malloc(colsB*sizeof(float));
for(i = 0; i < rowsA; i++){ //Matrix Mult algorithm
for(j = 0; j < colsB; j++){
sum = 0;
for(k = 0; k < colsA; k++)
sum+=(inpA[i][k]*inpB[k][j]);
outC[i][j] = sum;
}
}
printf("END\n");
printf("Result: %d x %d\n", rowsA, colsB); //print results to screen
for(i = 0; i < rowsA; i++){
for(j = 0; j < colsB; j++){
printf("%f ", outC[i][j]);
}
printf("\n");
}
if(fileC = fopen("outputC.txt", "w")){ //output to formatted file
fprintf(fileC, "%d %d\n", rowsA, colsB);

for(i = 0; i < rowsA; i++){
for(j = 0; j < colsB; j++){
fprintf(fileC, "%f ", outC[i][j]);
}
}
}
```

```
}
fclose(fileC); //close file stream

for(i=0;i<rowsB;i++)
free(inpB[i]); //free memory space
free(inpB);

for(i=0;i<rowsA;i++)
free(inpA[i]); //free memory space
free(inpA);

for(i=0;i<rowsA;i++)
free(outC[i]); //free memory space
free(outC);

printf("\n");
return 0;
}
```

□

2. Read the manual pages on the `open()` and `mmap()` function calls. You can either use "man open" or use the web to find information about these function calls. `mmap()` allows you to map the file contents to memory address spaces and then you can write to the memory addresses to write to the file and similarly reading from memory causes data to be read from the file.

Write a program using `open()` and `mmap()` to create a file containing integers from 1 to 100. Your program should write to file by writing to memory. Provide comments in your code.

The data in memory is in binary format and hence the data in the file will also be in binary.

```
Solution. #include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
```

```
#define FILEPATH "output.bin" //output file name
#define NUMBEROFINTS (100) //Number of integers to print to file
```

```
#define FILESIZE (NUMBEROFINTS * sizeof(int)) //size of file created based on NUMBEROFINTS

int main(int argc, char *argv[])
{
int i; //iterator used to write NUMBEROFINTS integers to file
int fp;
int result; //used for error checking
int *map; //array of ints using mmap()
//open file
fp = open(FILEPATH, O_RDWR | O_CREAT, (mode_t)0600);
if (fp == -1) {
perror("Error opening file for writing");
exit(EXIT_FAILURE);
}

//resize file to correct length for number ints
result = lseek(fp, FILESIZE-1, SEEK_SET);
if (result == -1) { //error check
close(fp);
printf("Error in resizing file\n");
exit(1);
}

//write an empty string to file
result = write(fp, "", 1);
if (result != 1) { //error check
close(fp);
printf("Error in writing last byte of the file\n");
exit(1);
}

//map file
map = mmap(0, FILESIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fp, 0);
if (map == MAP_FAILED) { //error check
close(fp);
printf("Error in mapping the file\n");
exit(1);
}

//write data to file/mapped array
```

```

for (i = 1; i <=NUMBEROFINTS; ++i) {
map[i] = i;
printf("%d\n", i);
}

//unmap the file
if (munmap(map, FILESIZE) == -1) { //error check
close(fp);
printf("Error in un-mmapping the file\n");
exit(1);
}
close(fp); //close file
return 0;
}

```

□

3. Construct a Verilog module for a controller. Your code must be written for synthesizability.

The controller has 3 states: *RESET*, *WAIT\_FOR\_ACK* and *FAIL*. The model starts up in *RESET*. When the signal *have\_data* is high, data is sent and the model moves to state *WAIT\_FOR\_ACK*. When *have\_data* is low, we wait in *RESET*. Once in *WAIT\_FOR\_ACK*, when *ack* is low, we continue in that state. If *ack* is high, we move to *RESET* again. If *have\_data* is high when in *WAIT\_FOR\_ACK*, we move to the *FAIL* state. Similarly we move to the *FAIL* state when *ack* is received in *RESET*.

Provide comments in your code.

*Solution.* The Verilog code for the controller is given below.

```

module controller(have_data, ack, state);
input have_data;
input ack;
output [1:0] state;

parameter RESET = 2'b00; // Define the state value for RESET
parameter WAIT = 2'b01; // Define the state value for WAIT
parameter FAIL = 2'b10; // Define the state value for FAIL

reg [1:0] state; //state is the output assigned in always block
// so it has to be declared as register

```

```
initial //initialize state to RESET state
state = RESET;

always@(ack or have_data) //Enter always block on an event on ack or have_data signal
begin
case(state) //Check the current state
RESET: //If the current state is RESET
if(have_data == 1 && ack != 1) //Check the input signals
state = WAIT; //next state
else if(ack == 1)
state = FAIL; //next state
else
state = RESET; //next state
WAIT: //if in WAIT...
if(ack == 1 && have_data != 1) //If the current state is WAIT
state = RESET; //next state
else if(have_data == 1)
state = FAIL; //next state
else
state = WAIT; //next state
FAIL: //If the current state is FAIL
begin
end
endcase
end
endmodule
```

□

4. Construct a Verilog module for a memory, along with a testbench for the same. Your model should have the following properties:
- The memory has 8 address bits as input, and is 4 bits wide.
  - There is an external *CLOCK* signal, such that upon any event on *CLOCK*, the data contained in the memory location referred to by the address is read from or written to the memory, if the state of the input *Read/Writebar* is 1 or 0 respectively.
  - The testbench must write a pattern of 0101 to each location of memory and read it back immediately. This should be done for each memory location. After this, the testbench must write

a pattern of 1010 to each location of memory and read it back immediately. This should be done for each memory location as well. If any of the read operations fails, the testbench should provide a message indicating failure, otherwise it should provide a message stating that the tests passed.

Provide comments in your code.

```
Solution. The code for memory module is given below:
/*This module implements 4-bits memory array of 256*/
/*dout is data output bus, din data input bus, addr is address bus*/
module memory(dout, din, CLOCK, addr, R_WBAR); //
output [3:0] dout; //output data
input [3:0] din; //input data
input [7:0] addr; //addr
input CLOCK, R_WBAR; //READ/WRITE control

reg [3:0] memory[0:255]; //256 element array to hold 4 bit data
reg [3:0] dout; //reg dout

always @ (CLOCK) //whenever there is an event on clock signal
begin
if(R_WBAR == 1) //read from memory location
dout =memory[addr];
else if(R_WBAR == 0) //write to memory location
memory[addr] = din;
end

endmodule
```

The code for testbench is as given below.

```
module testbench();
reg CLOCK, R_WBAR; //inputs to memory
reg [7:0] address; //address input
reg [3:0] datain; //data to be written to memory
wire [3:0] dataout; //data read from memory

integer i; //counter for "for-loop"

//instantiate memory module
```

```
memory MEM1(dataout, datain, CLOCK, address, R_WBAR);

initial //make our clock signal
begin
while(1)
begin
#10 CLOCK = ~CLOCK;
end
end

initial
begin //Write 0101 into a memory location and read it back
for(i = 0; i < 8'b11111111; i = i + 1)
begin
#10 address = i;
datain = 4'b0101;
R_WBAR = 0;
#10 R_WBAR = 1;
#10 if(dataout != 4'b0101) //if error
begin
$display("FAILED TO READ ADDRESS %b", address);
$finish;
end
end
#10
//Write 1010 into a memory location and read it back
for(i = 0; i < 8'b11111111; i = i + 1)
begin
#10 address = i;
datain = 4'b1010;
R_WBAR = 0;
#10 R_WBAR = 1;
#10 if(dataout != 4'b1010)
begin
$display("FAILED TO READ AT ADDRESS %b", address);
$finish;
end
end
#10 $display("MEMORY IMPLEMENTATION AND TESTING PASSED!");
```

```
#10 $finish;  
end
```

```
initial //To finish the program after so much time  
begin  
#20000 $finish;  
end  
endmodule
```

