Test Cases – Chapter 1

## Shell

This requirement has been met and can be demonstrated by loading the VM and seeing the prompt (VM> )

## load –d program.osx, memory model

This requirement has been met and can be demonstrated by running the following command:

Load -v CPUBound.osx

Result:

VM>load -v cpubound.osx

cpubound.osx

Program Size: 94

Program Start: 16

Load address: 100

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 1

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 16

Loading directive byte: 39

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 0

Loading directive byte: 0

Op-code: 0

Storing: 3,0

Op-code: 0

Storing: 5,8

Op-code: 4

Storing: 5,5

Op-code: 0

Storing: 0,12

Op-code: 4

Storing: 0,0

Op-code: 22

Storing: 2,0

Op-code: 0

Storing: 1,4

Op-code: 16

Storing: 2,2,1

Op-code: 2

Storing: 2,3

Op-code: 20

Storing: 2

Op-code: 12

Storing: 2,5

Op-code: 8

Storing: 58

Op-code: 7

Storing: 16

Op-code: 7

Storing: 0

## run –d program.osx, fetch-decode-execution

This requirement has been met and can be demonstrated by running the following:

run -v test\_cases.osx

VM>execute -v prog1.osx 10 CPUBound.osx 12

RESULT:

Loaded 312 bytes of data into main memory from prog1.osx starting at address 100

Header Information:

Program Size: 312

First Instruction Address: 112

Load Address: 100

PCB for: prog1.osx

====================================

PID: 103

Arrival Time: 10

Current State: stateNew

Main Memory

-----------

Load Address: 100

Program Size: 312

First Instruction: 112

PC: 112

Registers

---------

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 112

R8: 412

R9: 0

R10: 112

R11: 112

R12: 0

Loaded 100 bytes of data into main memory from CPUBound.osx starting at address 500

Header Information:

Program Size: 100

First Instruction Address: 516

Load Address: 500

PCB for: CPUBound.osx

====================================

PID: 104

Arrival Time: 12

Current State: stateNew

Main Memory

-----------

Load Address: 500

Program Size: 100

First Instruction: 516

PC: 516

Registers

---------

R0: 0

R1: 0

R2: 0

R3: 0

R4: 0

R5: 0

R6: 0

R7: 516

R8: 600

R9: 0

R10: 516

R11: 516

R12: 0

OP\_MVI: 3, 2

Setting R3 to: 2

OP\_MVI: 4, 6

Setting R4 to: 6

OP\_MVI: 5, 16

Setting R5 to: 16

OP\_MVI: 0, 28

Setting R0 to: 28

OP\_ADD: 2, 3, 4

Setting R2 to 2 + 6 = 8

R2 = 8

coredump –d program.osx

This requirement has been met and can be demonstrated by running the following:

Coredump CPUBound.osx

**Results:**

VM>coredump cpubound.osx

cpubound.osx

Directives

Directive : 0

Directive : 0

Directive : 0

Directive : 0

Directive : 1

Directive : 0

Directive : 0

Directive : 0

Directive : 16

Directive : 39

Directive : 0

Directive : 0

Directive : 0

Directive : 0

Directive : 0

Directive : 0

Instructions

Instruction : 0 : 0 : 0 : 0 : 0 : 0

Instruction : 0 : 3 : 0 : 0 : 0 : 0

Instruction : 0 : 5 : 8 : 0 : 0 : 0

Instruction : 4 : 5 : 5 : 0 : 0 : 0

Instruction : 0 : 0 : 12 : 0 : 0 : 0

Instruction : 4 : 0 : 0 : 0 : 0 : 0

Instruction : 22 : 2 : 0 : 0 : 0 : 0

Instruction : 0 : 1 : 4 : 0 : 0 : 0

Instruction : 16 : 2 : 2 : 1 : 0 : 0

Instruction : 2 : 2 : 3 : 0 : 0 : 0

Instruction : 20 : 2 : 0 : 0 : 0 : 0

Instruction : 12 : 2 : 5 : 0 : 0 : 0

Instruction : 8 : 58 : 0 : 0 : 0 : 0

Instruction : 7 : 16 : 0 : 0 : 0 : 0

## errordump –d program.osx

This requirement has been met and can be demonstrated by running the following:

Errordump test\_cases.osx

Results:

VM>errordump test\_cases.osx

SPACE HOLDER FOR ERROR DUMP

## Test Cases for each operation code in the assembler

This requirement has been met and can be demonstrated by running the following:

load test\_cases.osx

run test\_cases.osx

**Results breakdown:**

**Arithmetic**

LINE 8 ADD 2 + 6 = 8

LINE 10 SUB 16 - 6 = 10

LINE 12 MUL 6 \* 2 = 12

LINE 14 DIV 28 / 2 = 14

**Move Data**

LINE 16 MOV R2 R5 R2 now = 16

LINE 18 MVI Copy immediate value of 18 to R2 = 18

LINE 20 ADR Find address for label "LABEL1" and place in R2 = 36

LINE 23 STR Storing int: 2 in memory location [100][2]

LINE 24 LDR Loading what was stored in memory location [100][2] int R2 = 2

LINE 26 STRB Storing 1 byte from R0 and storing in memory location [100][2]

LINE 27 LDRB Loading 1 byte of what is stored in memory at [100][2] and storing in R2

**Branch**

LINE 29 B Jump to JUMP1

LINE 31 BL Jump to JUMP2 and store PC+6 in R2

LINE 36 BX Jump to JUMP3

LINE 42 BNE since zregister is != 0 jump to JUMP4

LINE 46 BEQ since zregister is = 0 jump to JUMP5

LINE 48 BGT since zregister is > 0 jump to JUMP6

LINE 51 BLT since zregister is < 0 jump to JUMP7

**Logical**

LINE 41 CMP check if 2 is equal to 3, setting the z register to != 0

Test Cases – Chapter 2

## Program requirement: execute –v Program\_1.osx <arriving time> Program\_2.osx <arriving time> … Program\_n.osx <arriving time>

**This requirement has been met and can be demonstrated with the following test cases:**

*Note on switches available*

*-v = Verbose - showing decode output*

*-g = Gantt chart, omitting verbose output and instead, showing <Clock cycle> <process ID> <PCB state> for each process that is loaded.*

* First demonstration shows that I am checking for memory overlap upon load, I have created an OSX file with an overlapping load location called badLoadAddress.osx

execute -g IOBound.osx 10 test\_cases.osx 13

Results:

Load address overlap exception.

Program ended with exit code: 1

* The second demonstration shows that when executing two or more programs, it will run them based on arrival time. To demonstrate this, run the first command and then the second command and notice that the order is different based on the arrival time modification.

execute -g prog1.osx 10 prog2.osx 12

Result: looking at the Gantt chart, you can see that prog1 ran first and prog2 ran second

execute -g prog1.osx 10 prog2.osx 8

Result: looking at the Gantt chart, you can see that prog2 ran first and prog1 ran second, this is because I adjusted the arrival time on prog2 to 8 which is lower than the arrival time of prog1 which is 10.

## Successfully implement 5 states model and PCB is updated

**This requirement has been met and can be demonstrated by using the -g switch which shows the different states. When running the following command, IOBound will require that you press 'q' to continue. After IOBound is complete, CPUBound will execute, which as a fork in it, so there will be a status of Wait from the parent shown.**

execute -g IOBound.osx 10 CPUBound.osx 12

Result: IOBound and CPUBound started in the "New" state, which can be seen at the very top where the PCBs are printed, IOBound then enters the running state while CPUBound enters the ready state, as shown in the gantt chart. After pressing 'q', IOBound enters the terminated state, then CPUBound enters the running state. CPUBound then forks into PID 0 and the parent PID, which is 102 in this case. The Parent enters the Waiting state while the child enters the running state. After the child terminates, the parent terminates as well. I was not able to get the final "Terminated" state for the parent to print out, but it did enter that state, which is why the program ended back the prompt.

## Successfully implement job queue, ready queue, I/O queue and PCB is updated

**This requirement has been met as demonstrated by the previous test case. Please let me know if there is something else you would like to see to demonstrate that this requirement has been met.**

## Successfully implement Wait system call and resources are released

**This requirement has been meet as seen by the parent entering the wait state while the child finishes. Once the child finishes execution, the child terminates and memory is cleaned up then the parent finishes execution. Demonstration from the Gantt chart when running:**

execute -g IOBound.osx 10 CPUBound.osx 12

## Successfully run osx under your shell

**This requirement has been met and can be demonstrated by running:**

osx IOBound.asm 800

Result:

osx version: 0.3

osX Assembly Complete see file IOBound.osx

## Successfully implement fork( ) – execute( )

**I have implemented fork() as demonstrated when executing CPUBound.osx. The fork actually loads a new copy of CPUBound into memory at the tail end of memory allocation. I am hoping this covers the requirement for exec() as well.**