CPSC 240: Computer Organization and Assembly Language
Assignment 03, Fall Semester 2024

CWID:__88461918____________ Name:__Sean DeFrank ________________

Quiz Questions:

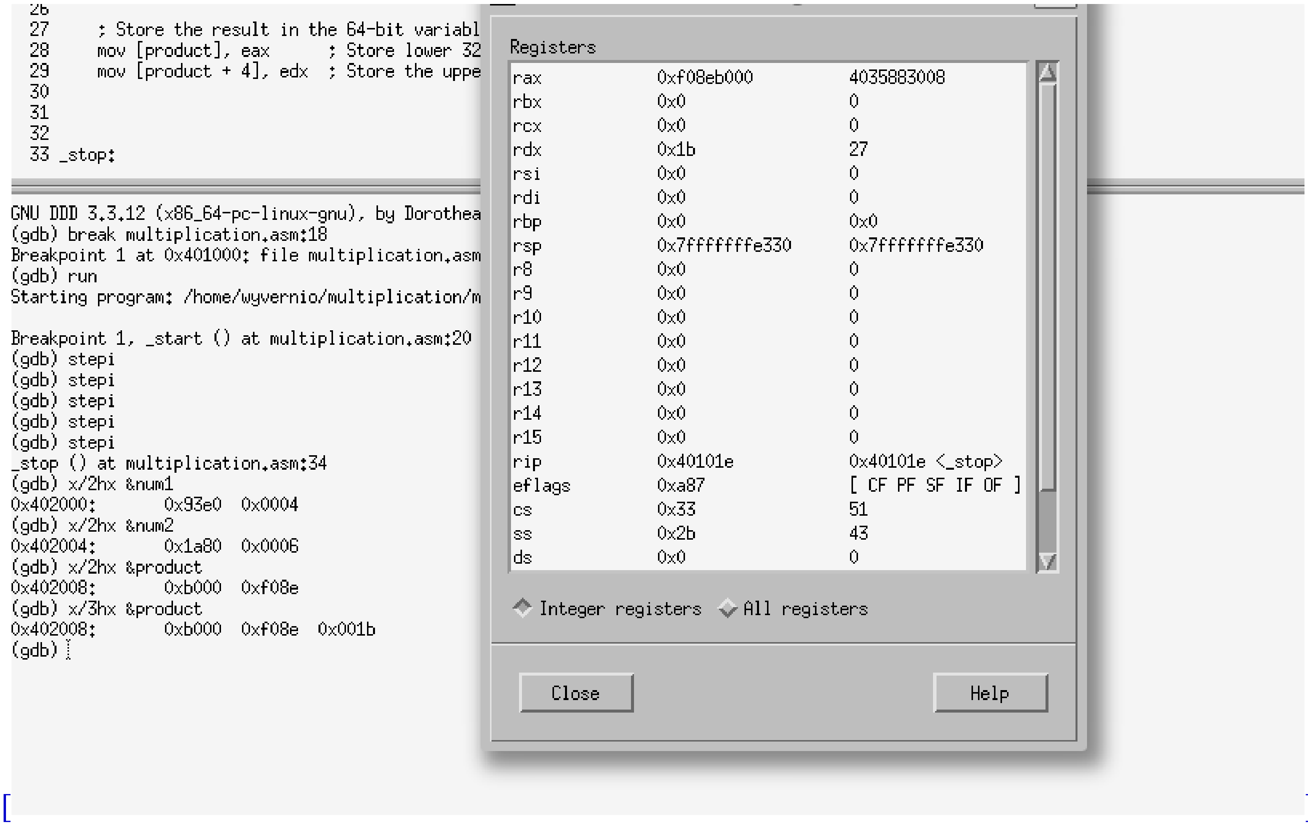
From the textbook "X86-64 Assembly Language Programming with Ubuntu," study quiz questions 13, 14, 15, and 16 on page 120 and page 121. Students do not need to submit answers to the quiz questions as they are found in Appendix D of the textbook.

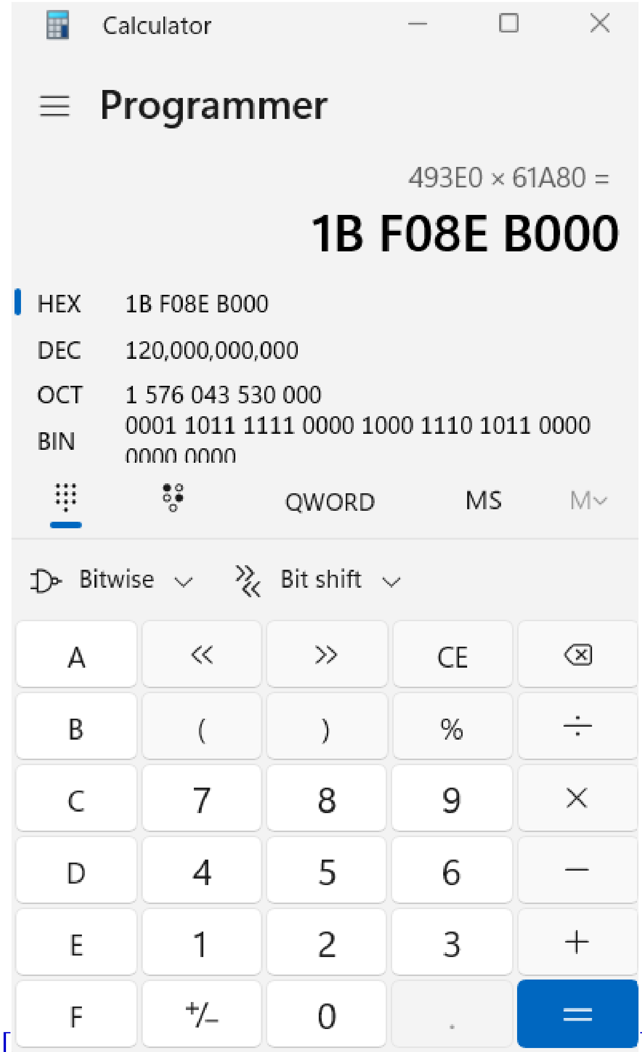
Programming:

1. Download the "CPSC-240 Assignment03.docx" document.
2. Design a 32-bit multiplication program "multiplication.asm", and use assembly language to realize the function of the following C++ instructions. NOTE: variable sizes and program functions should be equivalent to C/C++ instructions.
```
unsigned int num1 = 300,000;        // use dd to declare 32-bit variable
unsigned int num2 = 400,000;        // use dd to declare 32-bit variable
unsigned long product = 0;          // use dq to declare 64-bit variable
product = long(num1 * num2);
```
3. Assemble the "multiplication.asm" file and link the "multiplication.o" file to get the "multiplication" executable file.
4. Run the "multiplication" file with the GDB debugger to display the simulation results of num1 and num2, as well as the simulation results of product.
5. Insert source code (multiplication.asm) and simulation results (GDB panel) of the memory (num1, num2, and product) in the document. Use calculator or hand calculation to verify simulation results.
6. Design a 32-bit division program "division.asm", and use assembly language to realize the function of the following C++ instructions. NOTE: variable sizes and program functions should be equivalent to C/C++ instructions.
```
unsigned long num1 = 50,000,000,000;         // use dq to declare 64-bit variable
unsigned int num2 = 3,333,333;               // use dd to declare 32-bit variable
unsigned int quotient = 0, remainder = 0;    // use dd to declare 32-bit variable
quotient = num1 / num2;
remainder = num1 % num2;
```
7. Assemble the "division.asm" file and link the "division.o" file to get the "division" executable file.
8. Run the "division" file with the GDB debugger to display the simulation results of num1 and num2, as well as the simulation results of quotient and remainder.
9. Insert source code (division.asm) and simulation results (GDB panel) of the memory (num1, num2, quotient, and remainder) in the document. Use calculator or hand calculation to verify simulation results.
10. Save the file in pdf format and submit the pdf file to Canvas before the deadline.

```
[;unsigned int num1 = 300000
;unsigned int num2 = 400000
;unsigned long product = 0
;product = long(num1*num2)



section .data
	SYS_exit equ 60
	EXIT_SUCCESS equ 0
	num1 dd 300000			; 32-bit variable for num1
	num2 dd 400000			; 32-bit variable for num2
	product dq 0			; 64-bit variable

section .text
	global _start			; Start

_start:
	; Load num1 and num2 into registers
	mov eax, [num1]			; Moving num1 into eax
	mov edx, [num2]			; Moving num2 into edx

	; Perform the multiplication
	mul edx				; Multiply eax by edx
					; The result is a 64-bit value stored in edx:eax

	; Result goes into 64-bit variable
	mov [product], eax		; Store lower 32-bit into eax
	mov [product + 4], edx	; Store the upper 32-bit into edx



_stop:
	mov	rax, SYS_exit		;terminate excuting process
	mov	rdi, EXIT_SUCCESS		;exit status
	syscall				;calling system services]
```

I decided to add the C++ code at the top of the assembly code commented in order to help me program, it let me visualize it better, just wanted to add a little addendum.

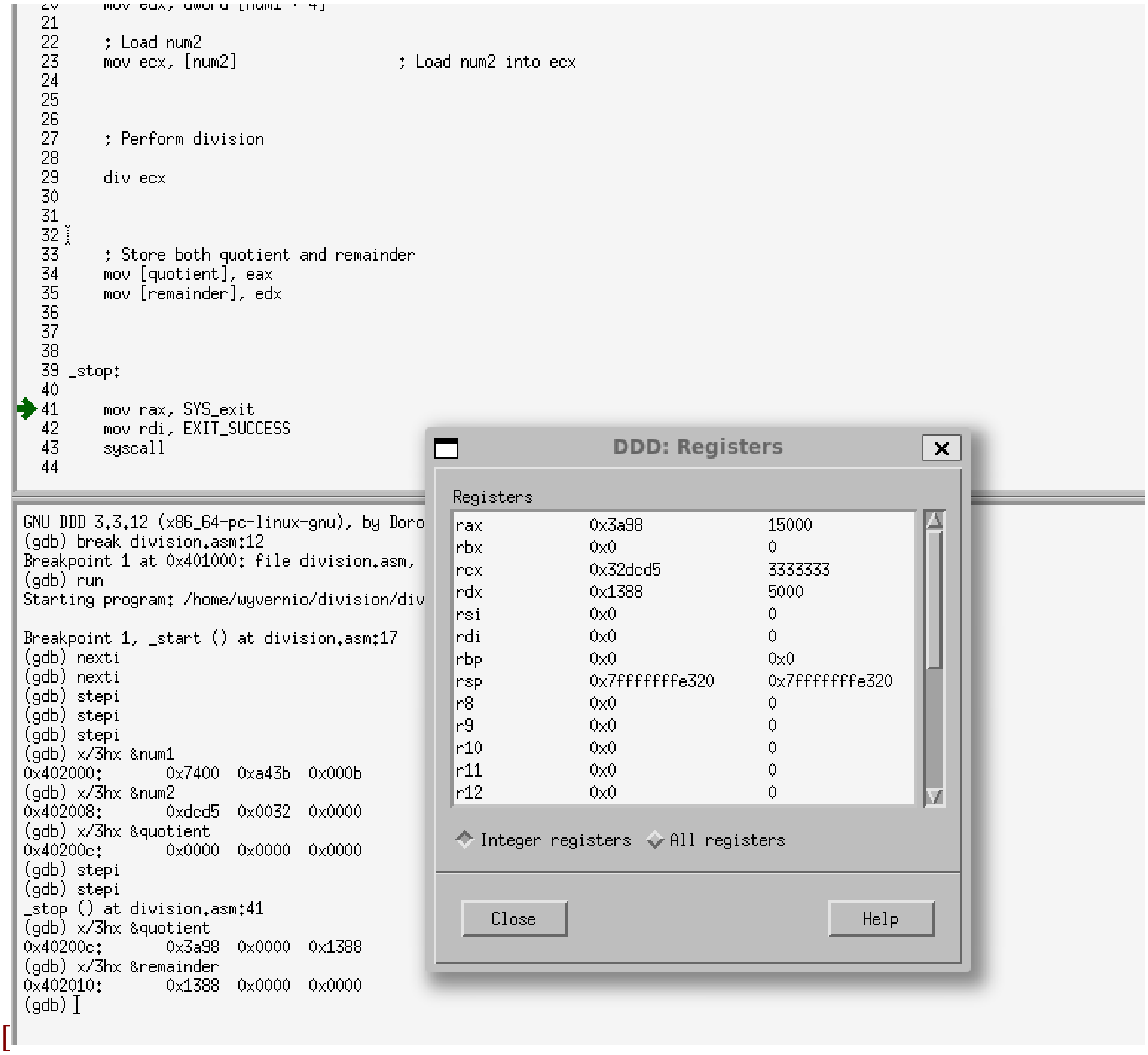

26
27 ; Store the result in the 64-bit variabl
28 mov [product], eax ; Store lower 32
29 mov [product + 4], edx ; Store the uppe
30
31
32
33 _stop:
GNU DDD 3.3.12 (x86_64-pc-linux-gnu), by Dorothea
(gdb) break multiplication.asm:18
Breakpoint 1 at 0x401000: file multiplication.asm
(gdb) run
Starting program: /home/wyvernio/multiplication/m
Breakpoint 1, _start () at multiplication.asm:20
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
_stop () at multiplication.asm:34
(gdb) x/2hx &num1
0x402000: 0x93e0 0x0004
(gdb) x/2hx &num2
0x402004: 0x1a80 0x0006
(gdb) x/2hx &product
0x402008: 0xb000 0xf08e
(gdb) x/3hx &product
0x402008: 0xb000 0xf08e 0x001b
(gdb)
Registers
rax 0xf08eb000 4035883008
rbx 0x0 0
rcx 0x0 0
rdx 0x1b 27
rsi 0x0 0
rdi 0x0 0
rbp 0x0 0x0
rsp 0x7fffffffe330 0x7fffffffe330
r8 0x0 0
r9 0x0 0
r10 0x0 0
r11 0x0 0
r12 0x0 0
r13 0x0 0
r14 0x0 0
r15 0x0 0
rip 0x40101e 0x40101e <_stop>
eflags 0xa87 [CF PF SF IF OF]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
Integer registers
All registers
Close
Help

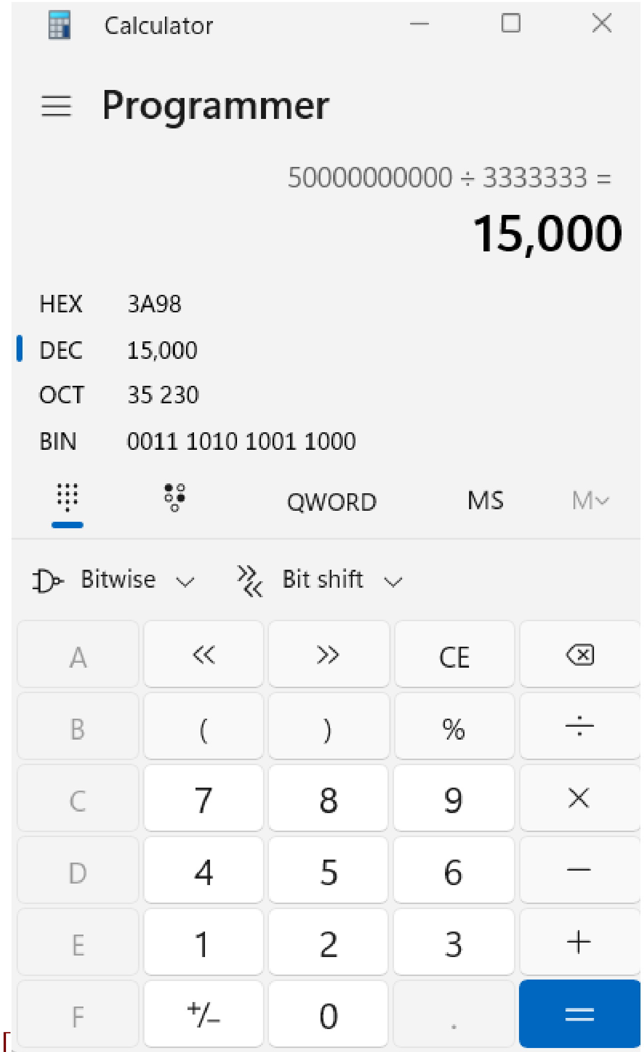
[
]

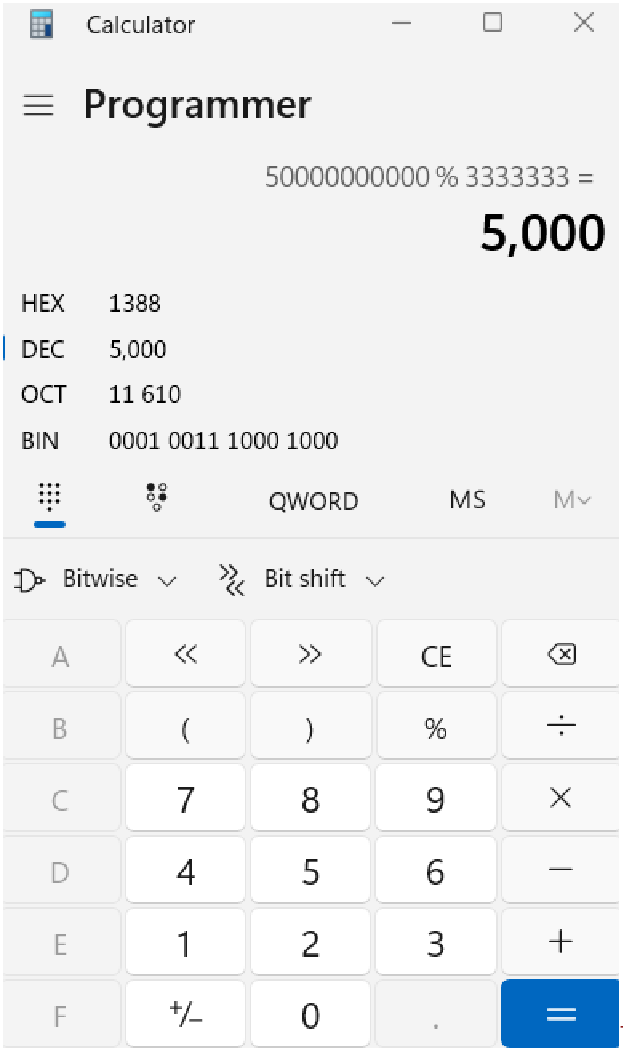
The result of the calculator is the same as my register, but backwards, which I'm told is normal, 1bf08eb000, which is acquired by multiplying 0x493e0 and 0x61a80, or 300k and 400k respectively.

```
[section .data
    SYS_exit equ 60
    EXIT_SUCCESS equ 0
    num1 dq 50000000000
    num2 dd 3333333
    quotient dd 0
    remainder dd 0

section .text
    global _start

_start:
    ; Load num1 into rdx:rax
```

```
    xor rdx, rdx

    mov eax, dword [num1]
    mov edx, dword [num1 + 4]

    ; Load num2
    mov ecx, [num2]                ; Load num2 into ecx



    ; Perform division

    div ecx



    ; Store both quotient and remainder
    mov [quotient], eax
    mov [remainder], edx



_stop:

    mov rax, SYS_exit
    mov rdi, EXIT_SUCCESS
    syscall                        ]
```


21
22 ; Load num2
23 mov ecx, [num2] ; Load num2 into ecx
24
25
26
27 ; Perform division
28
29 div ecx
30
31
32
33 ; Store both quotient and remainder
34 mov [quotient], eax
35 mov [remainder], edx
36
37
38
39 _stop:
40
41 mov rax, SYS_exit
42 mov rdi, EXIT_SUCCESS
43 syscall
44
GNU DDD 3.3.12 (x86_64-pc-linux-gnu), by Doro
(gdb) break division.asm:12
Breakpoint 1 at 0x401000: file division.asm,
(gdb) run
Starting program: /home/wyvernio/division/div
Breakpoint 1, _start () at division.asm:17
(gdb) nexti
(gdb) nexti
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) x/3hx &num1
0x402000: 0x7400 0xa43b 0x000b
(gdb) x/3hx &num2
0x402008: 0xdcd5 0x0032 0x0000
(gdb) x/3hx &quotient
0x40200c: 0x0000 0x0000 0x0000
(gdb) stepi
(gdb) stepi
_stop () at division.asm:41
(gdb) x/3hx &quotient
0x40200c: 0x3a98 0x0000 0x1388
(gdb) x/3hx &remainder
0x402010: 0x1388 0x0000 0x0000
(gdb)
DDD: Registers
Registers
rax 0x3a98 15000
rbx 0x0 0
rcx 0x32dcd5 3333333
rdx 0x1388 5000
rsi 0x0 0
rdi 0x0 0
rbp 0x0 0x0
rsp 0x7fffffffe320 0x7fffffffe320
r8 0x0 0
r9 0x0 0
r10 0x0 0
r11 0x0 0
r12 0x0 0
Integer registers
All registers
Close
Help


Calculator
Programmer
50000000000 ÷ 3333333 =
15,000
HEX 3A98
DEC 15,000
OCT 35 230
BIN 0011 1010 1001 1000
QWORD
MS
M
Bitwise
Bit shift
A
CE
B
(
)
%
C
7
8
9
D
4
5
6
E
1
2
3
F
+/-
0
.
=



]

The quotient would be 15k while the remainder would be 5k, proven by the calculator and shown in the register. I decided to show this one using decimal so it would be a little easier to type, but you can still see the hex codes on the left. Have a good day, professor. 😊