Midterm Review

CPSC 240-09

May 7, 2019

26. In a program, how are these sections used? Of three types of memory configurations (read-write, read-only, and executable), which apply to which section? (note: one section may have two types)

A .data

Used to allocate variables that we have pre-initialized to some value.

Read-write (because we can read and write memory variables in this space)

B .text

Used to put our instructions in.

Executable because we want to be able to execute instructions that we have in this section.

Read-only because we don't want to accidentally write over existing instructions (it could make the processor try to execute weird things).

C .bss

Used to allocate variables in that do not have a pre-initialized value.

Read-write (because we can read and write memory variables in this space)

27. **Given this sequence of bytes in memory (in an x86 64-bit computer), what would it look like when moved to a 64-bit register with an instruction like this: mov rax, tempVar**

    tempvar db 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0xDE, 0xF0

rax would look like: F0DEBC9A78563412

28. **The cdqe instruction can be used to convert a signed number in eax to a signed number in rax. If -65 is in eax, which looks like: FFFFFFBF. What does the rax register have in it after the cdqe instruction is executed?**

Answer: The sign is extended to "higher" bits. So, if the sign bit of eax is 1, then all the higher bits will be one and the answer would be FFFFFFFFFFFFFFBF

If the sign bit of eax is 0, then zeros would be in the higher bits. Say, eax contained +65 (00000041), then rax would contain 0000000000000041

29. **In a 64-bit mode x86 program, the register names rbx, ebx, bx and bl all refer to the same register. What is the difference between them?**

rbx refers to the full 64 bits

ebx refers to the lower 32 bits of rbx

bx refers to the lower 16 bits of rbx

bl refers to the lower 8 bits of rbx

64-bit register	Lowest 32-bits	Lowest 16-bits	Lowest 8-bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Provide the equivalent assembly language instructions for the following C language statements – assume all variables are 64-bit:

30. qSum = qNum1 + qNum2;

```
mov   rax, qword [qNum1]
add   rax, qword [qNum2]
mov   qword [qSum] , rax
```

31. qSum = qNum1 - qNum2;

```
mov   rax, qword [qNum1]
sub   rax, qword [qNum2]
mov   qword [qSum] , rax
```

32. qSum = qNum1 * qNum2;

```
mov   rax, qword [qNum1]
mul   qword [qNum2]          ; mul hardcoded to use rax
mov   qword [qSum] , rax
```

Alternatively, you could use imul:

```
mov   rax, qword [qNum1]
imul  rax, qword [qNum2]     ; There are three ways to use imul
mov   qword [qSum] , rax
```

33. qSum = qNum1 / qNum2;

```
mov   rax, qword [qNum1]
cqo                          ; cqo expands number in rax to rdx:rax
div   rax, qword [qNum2]
mov   qword [qSum] , rax
```

34. qSum = qNum1 % qNum2;

```
mov   rax, qword [qNum1]
cqo                          ; cqo expands number in rax to rdx:rax
div   rax, qword [qNum2]
mov   qword [qSum] , rdx     ; Save the remainder.
```

35. Where are the flags `CF`, `AF`, `ZF`, `OF`, `SF`? What is the purpose of the ZF, CF and SF flags.

ZF – Zero flag indicates the result (from compare) is zero or equal

CF – Carry flag indicates the arithmetic operation caused a carry or unsigned comparison results in below

SF – Sign flag is used in signed comparisons (along with the OF flag)

(SIDE NOTE: A compare sets flags by subtracting the second operand from the first operand, except it does not save the result)

36. List all of the general purpose registers in the x86 architecture when in 64-bit mode?

rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp, r8,r9, r10, r11, r12, r13, r14, r15

37. What are four instructions that use the stack pointer?

push, pop, call, ret

How do each of the four instructions from the previous question work and how is the stack used?

38. push will push some value onto the stack (either 64 bit variable or register)

39. pop will pop off some value from the stack (to either 64 bit variable or register)

40. call will push the value in rip onto the stack and the jump to a label

41. ret will pop off the value from the stack and put it into rip

42. What is the difference between sar and shr instructions?

sar - Shift right arithmetic – Shifts to the right but sign bit always stays the same

shr – Shift right logical – shifts to the right and always shifts in zero

43. What does instruction "xor rdi, rdi" do?

We can use xor to clear a register. This instruction clears rdi to 0.

44. What is a label and does it have a size in an assembly program when using nasm?

A label provides an address for a jump or call instruction. When using a jump or call instruction and the label name, the next instruction to execute will be at the label's address.

45. What is the difference between the je and jz instructions?

They are the same instruction.

46. What is the difference between the jb and jl instructions?

Jb (jump if below) is used after a compare when intending it to be an unsigned comparison. Jl (jump less) is used after a compare when intending it to be a signed comparison.

47. What type of file(s) is/are outputted by nasm when assembling a program that has no errors if you use the –l option?

.o file for object output and a listing file

48. What symbol marks the program entry point in an assembly program, assuming there are no C/C++ libraries being used? Who needs to know it and why?

_start the linker needs to know it

49. What type of file(s) are inputted to the linker (ld)? What type of file is outputted by the linker?

A .o file (from the output of the assembler) is an input file. An executable file is the output from the linker.

50. Which registers are used, and for what purpose, for a read system call (using syscall to interface to the Linux)?

```
mov     rax, 0          ; System call for read
mov     rdi, qword [fileDesc]    ; File handle from Open
mov     rsi, readBuffer          ; Address of read buffer
mov     rdx, readBufferLen       ; Size of message in # of bytes
syscall                 ; invoke operating system to do the write
```

Linux has one order of registers used to pass parameters: rax, rdi, rsi, rdx (See Appendix C in the book on page 315).

ABI (Application Binary Interface Specification for AMD64 – 64-bit) has another order of registers used to pass parameters: rsi, rdi, rdx, rcx, r8, r9 (See Chapter 12 page 172 and 173).

51. How is the rbp register used?

To set up a stack frame.

52. In the following code, what is in the variable "test" after each of the mov instructions? (Notice what is in "test" to begin with!)

```
Section .data
test: dq -1

Section .text

mov byte[test], 1  ;1
mov word[test], 2  ;2
mov dword[test], 3 ;4
mov qword[test], 4 ;8
```

test:	FF	FF	FF	FF	FF	FF	FF	FF

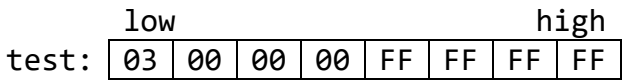
53. In the following code, what is in the variable "test" after each of the mov instructions? (Notice what is in "test" to begin with!)

```
section .data
test: dq -1

section .text

mov byte[test], 1  ;1
mov word[test], 2  ;2
mov dword[test], 3 ;4
mov qword[test], 4 ;8
```

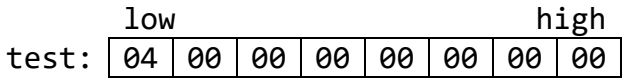
After: `mov byte[test], 1  ;1`

After: `mov word[test], 2  ;2`

After: `mov dword[test], 3  ;4`



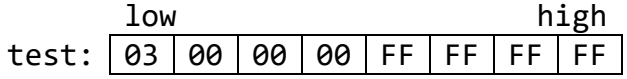
After: `mov qword[test], 4  ;8`



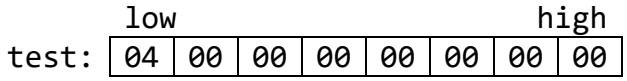
After: mov byte[test], 1 ;1 ➔ FF FF FF FF FF FF FF 01

After: mov word[test], 2 ;2 ➔ FF FF FF FF FF FF 00 02

After: mov dword[test], 3 ;4 ➔ FF FF FF FF 00 00 00 03



After: mov qword[test], 4 ;8 ➔ 00 00 00 00 00 00 00 04



54. **Please write a short subroutine that translates a number between 0-15 passed in al to its ASCII hexadecimal equivalent (similar to our xlat assignment, but just a subroutine). Remember to save/restore any registers you may be using in the subroutine**.

What is a subroutine (aka function)?

A piece of code that is called using the call instruction whose caller expects to get control back after the subroutine has completed.

```
sub1:
     push  rbp
     mov   rbp, rsp
     …
     …
     pop   rbp
     ret
```

Problem # 54 minimum answer:

```
; Function to return a single ascii code
; hex character (0-9,A-F) given an integer
; between 0-15 in al. ascii code is returned in al
gethexascii:
     push  rbx             ; Save rbx because we are using it
     mov   rbx, hexascii
     xlat
     pop   rbx             ; Restore rbx before returning
     ret

hexascii db "0123456789ABCDEF"
```

Does hexascii need to be in the .data section or can it be in the .text section?

Write the equivalent assembly code for the following high-level (c/c++) constructs (short snippets; all variables are quad-words in size, but do not include variable definitions in your code):

55.

```
if ( qRetCode >= 0) {
     qFileDesc = qRetCode;
} else {
     qFileDesc = -1;
}
```

```
     cmp   qword [qRetCode], 0
     jl    else
     mov   rax, qword [qRetCode]
     mov   qword [qFileDesc], rax
     jmp   aroundElse
else:
     mov   qword [qFileDesc], -1
aroundElse:
```

Write the equivalent assembly code for the following high-level (c/c++) constructs (short snippets; all variables are quad-words in size, but do not include variable definitions in your code):

56.

```
While ( qCount != 0) {
     qCount = qCount – 1;
}
```

```
loop:
     cmp   qword [qCount], 0
     jl    break

     sub   qword [qCount], 1
     jmp   loop

break:
```

Write the equivalent assembly code for the following high-level (c/c++) constructs (short snippets; all variables are quad-words in size, but do not include variable definitions in your code):

57.

```
for ( i = 0; i < 10; i++) {
      if ( i > 2 )
      qSum = qSum + i;
}
```

```
      mov   qword[i], 0
loop:
      cmp   qword [i], 10
      jge   break

      cmp   qword [i], 2          ; if ( i > 2 )
      jle   around
      mov   rax, qword [i]        ; qSum = qSum + i
      mov   qword [qSim], rax
around:

      inc   qword [i]
      jmp   loop
break:
```