Basic Concepts Worksheet #1: Basic Concepts

True-False

1. The binary representation of decimal 42 is 00101010.
   - True

2. The hexadecimal representation of decimal 35 is 33h
   - False. It is 0x35 or 35h

3. The sum of the binary integers 01101101 and 00111011 is 10101001.
   - False. It is 010101000

4. The 8-bit two's complement of binary 00000010 is 11111110.
   - True. Invert bits, then add one

5. The binary representation of decimal -42 is 11010111.
   - False. It is -0b101010

6. Suppose there is a virtual machine containing levels V1 and V2, where V2 is above V1 in the machine
hierarchy. The programs written in language V2 can be executed by a program running at level V 1.
   - False. There is no interpreter for the V2 language so it cannot be run. It is a higher level language, much like C to assembler.

7. The sum of all powers of 2 from 2^0 to 2^8 is 511.
   - True

8. A virtual machine may be constructed from software.
   - True. VMWare, virtualbox, etc.
9. The sum of 3AB4h and 0429h is 3EDDh.
   - True
10. To translate an unsigned decimal integer into binary, repeatedly divide the integer by 2, saving each
remainder as a binary digit.
   - True. This also works for reversing the number. For example, you give the program 54 and it prints 45.
11. The expression !X && !Y is false when X is false and Y is true.
   - True

Short Answers

1. What are the hexadecimal and decimal representations of the ASCII character capital B?
   - 0x42
   - 66
2. What are the hexadecimal and decimal representations of the ASCII character capital G?
   - 0x47
   - 71
3. The following 16-bit hexadecimal numbers represent signed integers. Convert each to decimal:
   a. 6BF9
      - 27641

   b. C123
      - 49443
4. What is the hexadecimal representation of each of the following binary numbers?
   a. 0011 0101 1101 1010
      - 35DA
   b. 1100 1110 1010 0011
      - CEA3
   c. 1111 1110 1101 1011
      - FEDB

Worksheet #2: First Look

True/False Section

1. In a computer system, a clock provides a way to synchronized different things that are happening in the computer.
	- True

2. In a computer system, the clock also keeps the time of day.
	- False

3. In most computer systems, memory is composed of lots of â\200\234bitsâ\200\235 that can remember data. Each bit can remember the number 0, 1, or 2.
	- False. 0 and 1 only

4. The type of memory that holds a program when it is being executed by the computer is generally called the primary memory or primary storage, and it is also called RAM (random access memory).
	- True

5. ERAM is a type of random access memory that is extra fast.
	- False

6. Secondary storage is another type of memory that is faster than primary storage, but cost more money, so there is less of it in a computer.
	- False

7. In a modified Harvard architecture, certain areas of memory can be configured as read-only, executable, and/or read-write.
	- True

8. In a 5-stage instruction execution cycle, the fetch stage is used to fetch an instruction to execute.
	- False

9. In a 5-stage instruction execution cycle, the fetch operands stage is used to fetch anything that the instruction might need from memory (that is, any operands needed by the instruction).
	- True

10. In main memory, in virtually all current modern processing architectures, each byte in memory has an address.
	- True

11. Theoretically, you could design a computer architecture where only each double word (the equivalent of 4 bytes) is addressable.
	- True

12. In the x86-64 architecture, there are 16 general purpose registers that are 64-bit in size.
	- True

13. If I just want to access the lowest byte of the RAX register, I can reference AL in instructions.
	- True

14. If I just want to access the lowest byte of the RDI register, I can reference DIL in instructions.
	- True

15. If I just want to access the lowest byte of the RFLAGS register, I can reference RFL in in
structions.
- False

16. The RIP register has the address of the last instruction that caused the computer to die.
- False

17. The address bus (or the address portion of the system bus) allow the memory to tell the pr
ocess the
address of the memory that was just read.
- True

18. In an x86 assembly program, the instruction can be either an actual x86 instruction, a pse
udo
instruction, a directive, or the name of a register.
- False

19. Instructions always has at least one operand.
- False. The xlat instruction does not require an operand

20. You can use either the ; (semicolon) or # (hash tag) to start a comment.
- True

21. In little-endianness, the memory address of a given value in memory, whether taken as a by
te,
word, double word, or quad word, is the same.
- True

22. There is only one set of page tables used on an x86-64 computer.
- False

23. A page fault tells the OS that a page is not in physical memory.
- False

24. Page tables for a specific process must be in memory while a program is running in that pr
ocess.
- True

Short Answer

1. What is the difference between these three sections:
a. .data
- Holds label declarations and other predefined data needed for program flow
b. .text
- The instructions the program will execute.
c. .bss
- Uninitialized data is stored here. Variables are declared but have no conten
t

2. What is the decimal equivalent of these byte sized binary numbers?
a. 10000000
- Unsigned: 128
- Signed: -128
b. 11111111
- Unsigned: 255
- Signed: -1

3. In your own words, give a definition of little-endianness.
- LSB is first. If DEADBEEF is stored, it will be stored as DEADBEEF where in big endi
an it would be stored as FEEBDAED
4. On a 64-bit Linux system (how big is the page size?) How many pages are available on a comp
uter
that has 8 Gigabytes of memory?
- 4 GB per page. Therefore we have 2 pages. This information may vary, run ‘getconf PA

GESIZE‘ and the page size is given in bytes.

5. What is the difference between the EIP register and the RIP register?
   - EIP register is the 32bit version of the RIP register

6. What is the name of the 32-bit register equivalent of the R8 register?
   - r8d

Jared Dyreson
CPSC-240 09
TR at 11:30 to 13:20
February 19, 2019

Short Answer

1. Assembler code representation for the C equivalent
    a. sum = number + other
        i. mov r10, 10
        ii. mov r11, 11
        iii. add r10, r11
        iv. ; add <dst> <src>
    b. subtrahend = number - other
        i. mov r10, 11
        ii. mov r11, 10
        iii. sub r11, r10
    c. quotient = dividend / divisor
        i. mov al, 10 ; dividend
        ii. mov ah, 0 ; remainder
        iii. mov bl, 2 ; divisor
        iv. div bl ; dividend / divisor
        v. mov byte[rdi], bl
        vi. mov rax, SYS_WRITE
        vii. mov rdx, 1
        viii. syscall
    d. modulo = dividend % divisor
        i. mov al, 10 ; dividend
        ii. mov ah, 0; remainder
        iii. mov bl, 2 ; divisor
        iv. div bl ; dividend / divisor
        v. mov byte[rdi], ah
        vi. mov rax, SYS_WRITE
        vii. mov rdx, 1

viii. syscall

2. Assembler flags and their meaning
    a. **CF: Carry Flag** → It indicates when an arithmetic carry or borrow has been generated out of the four most significant bits
        i. 101 + 101 = will result in a carry flag when you add the last elements together
    b. **AF: Auxiliary Flag** → It indicates when an arithmetic carry or borrow has been generated out of the four least significant bits
    c. **ZF: Zero Flag** → to check the result of an arithmetic operation, including bitwise logical instructions. It is set to 1, or true, if an arithmetic result is zero, and reset otherwise
    d. **OF: Overflow Flag** → a single bit in a system status register used to indicate when an arithmetic overflow has occurred in an operation, indicating that the signed two's-complement result would not fit in the number of bits used for the operation
    e. **SF: Sign Flag** → a single bit in a system status (flag) register used to indicate whether the result of the last mathematical operation resulted in a value in which the most significant bit was set
    f. These registers are stored in the **eflags** register
3. The general purpose registers in x86-64 ASM
    a. r{8..15} : allow for the use of putting anything in them and not possibly messing up program flow (generally)
    b. rax : place syscalls here
    c. Rsi : content of SYS_WRITE
    d. Rdi : STDOUT Flag
    e. Rdx : strlen(message)
4. Four instructions that use the stack pointer and their purpose
    a. Push : push a value onto the stack
    b. Pop : take a value off the stack
    c. Pushf : save the status register to the stack
    d. Popf : restore the status register from the stack
5. Difference between **sar** and **shr**
    a. **shr** : Performs a logical shift right operation on destination operand

    b. **sar** : Performs an arithmetic shift right operation on destination operand
6. This operation “xor rdi, rdi” will clear the rdi register without having to do a mov instruction
7. The connection to these registers is that they are on the same register, rax being the full 64 bit lane, eax 32, ax 16, al 8. Each are special in their own right.
8. The **rip** register points to the next instruction to be executed
9. A label is a ghetto version of a function that can be treated as a section of code to be either executed in sequence or on a conditional basis, depending on program flow. It does not appear to have a size.
10. **je** is “jump if the compare was equal to a constant other than 0”. **jz** is “jump if zero”
11. **jb** “jump below” and **jl** “jump if less”
12. Nasm will spit out an unlinked executable and is useless unless linked by **ld**, which will then render a coherent ELF executable (depending on architecture)
13. Symbols are used to debug and are used by GDB to move through program flow. They are a roadmap for GDB.
14. Linker (ld) will produce an ELF executable or more verbosely put:
    a. ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
15. Consider the following code
    a. mov rax, SYS_READ ; rax holds our syscall for reading in file
    b. mov rdx, BUFFER_SIZE; we tell rdx the size of the buffer
    c. mov rdi, 1; indicates how big our data type is (sizeof(char))
    d. syscall
16. Those values provided in the above code snippet are then issued to the kernel, requesting permission to read the file
17. The stack frame is a data structure and more specifically the place where all programs live. Values that are loaded in first are let out last. This is especially helpful when we want the base pointer to be buried at the bottom so it does not get accidentally popped off. The reason why recursion can exist
18. The **rbp** is the base pointer of the stack, best kept at the bottom.
19. The values go as follows
    a. -255, -65535, 1, 1

20. Dissecting the r9 register
    a. r9 → 64
    b. r9d → 32
    c. r9w → 16
    d. r9b → 8
21. You can test rdx by running the **test** operand
22. Subroutine to translate numbers to hex
    a. See last page

```
; convert hex to ascii using offset values
; Jared Dyreson
; CPSC-240-09 TR @ 11:30 - 13:20

; some light -> https://stackoverflow.com/questions/36336045/print-register-value-to-console

; I also included a bash script that I was using to compile the ASM files, thought it might be
 useful for the class
; cmp function -> https://stackoverflow.com/questions/45898438/understanding-cmp-instruction
; jmp with conditions -> https://stackoverflow.com/questions/1123396/assembly-to-compare-two-n
umbers
; ^ https://en.wikibooks.org/wiki/X86_Assembly/Control_Flow
global _start

SECTION .text

_start:
  cmp byte[bval], 16 ; if a >= b; then move to the exit function
  jge _exit
  mov rdi, 1
  mov al, byte[bval]
  lea rbx, [xtable]
  xlat
  mov byte[bval], al
  mov rsi, bval
  xor rax, rax
  mov rax, 1
  mov rdx, 2 ; if you pass in length, it prints the content of the a.out bin for some odd, ine
xplicable reason
  syscall
  jmp _exit
_exit:
  mov rax, 60
  mov rdi, 0
  syscall
SECTION .data

length: equ $bval
xtable: db '0123456789ABCDEF', 10
bval: db 15, 10
```