No Pages:22

MICROPROCESSOR & AND ASSEMBLY LANGUAGE

Spring 2020 Mid Term

(Solution)

Student Name: Ashir Ali Khan

ID#:11533

Degree: BSCS(Batch09)

Course Codes: CSC-304

EDP Codes: 102002094

Instructor: Muhammad Amin

Duration: 13th April to 18th April

Department Of Computer Science

Ashis Ali Khan
ID # 11533
BSCS #09

Coursc/S# CSC-304
Instructor Mr. Muhammad Amin

Q1. Solve the following

1/(a) $64_{10} = (?)_2$

2	64
2	32 – 0
2	16 – 0
2	8 – 0
2	4 – 0
2	2 – 0
	1 – 0

$64_{10} = (1000000)_2$

1/(b) $(01111111)_2 = (?)_{10}$

$= (0 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

$= 0 + 64 + 32 + 16 + 8 + 4 + 2 + 1$

$= 127$

1/(c) $4D7F_{16} = (?)_{10}$

$= (4 \times 16^3) + (13 \times 16^2) + (7 \times 16^1) + (15 \times 16^0)$

$= (4 \times 4096) + (13 \times 256) + (7 \times 16) + (15 \times 1)$

$= 16384 + 3328 + 112 + 15$

$= 19,839$

1/(d) $128_{10} = (?)_{16}$

16	128
	8 – 0

$128_{10} = (80)_{16}$

Q1 part(e)

$3A6F_{16} = (?)$

3	A	6	F
0011	1010	0110	1111

$(0011\,1010\,0110\,1111)_2$

Q1 part(f)

$1100\,0011\,1100\,0101_2 = (?)_{16}$

1100	0011	1110	0101
C	3	E	5

$(C3E5)_{16}$

Q1 part(g)

$11111111_2 = \pm(?)_{10}$

MSB is 1 shows the number is negative

$-(1\times2^7) + (1\times2^6) + (1\times2^5) + (1\times2^4) + (1\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0)$

$= -128 + (64+32+16+8+4+2+1)$

$= -128 + 127$

$= -1$

Q1 part(h)

$-16_{10} = (?)_2$

11533

Q1 Part(h)

$(-16)_{10} = (?)_2$

2	16
2	8 – 0
2	4 – 0
2	2 – 0
	1 – 0

$(00010000)_2$

Taking 2's compliment

00010000

11101111

+ 1

$(11110000)_2$ Ans.

Q1 Part(i)

$(01111111)_2 - (00000111)_2$

01111111

− 00000111

01111000

Taking 2's compliment

10000111

+1

$(10001000)_2$ Ans.

11533

Q1 part(j)

$6D_{16} - 3f_{16}$

6 D) - 3 F

0110 1101 0011 1111

6D = 0110 1101

3F = 0011 1111

Take first number of second number.

0111111

↳ 1000000

Adding both number

1101101
1000000

carry (1) 0101101

↳

0101101
+1

0010 1110

2 E

(2E)16 Ans

Question #02

Part a

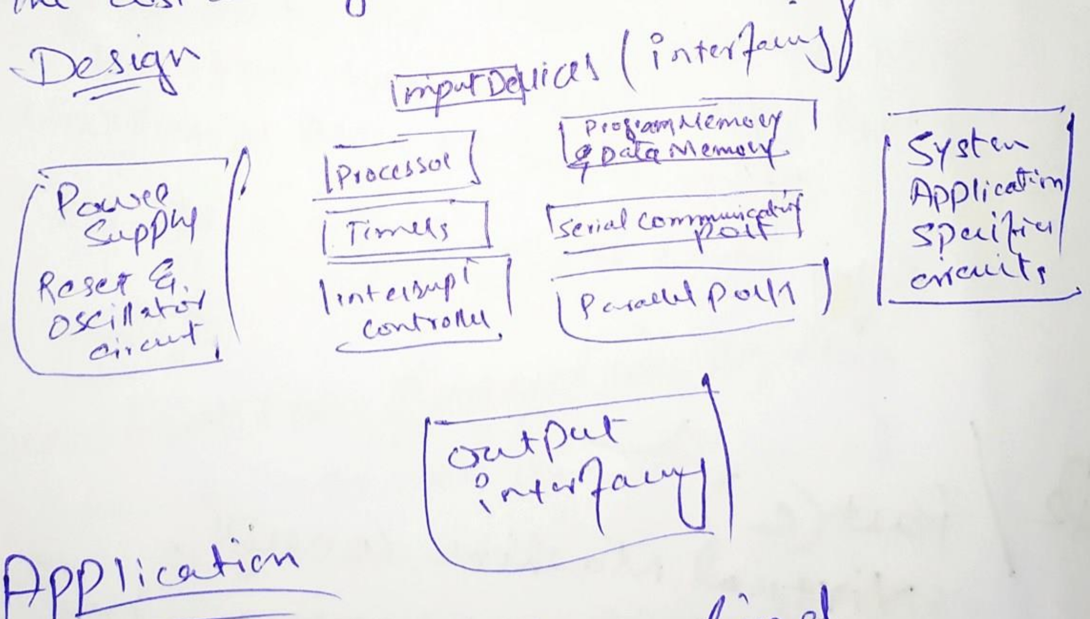
Embedded Systems

The electronic system which integrates the hardware circuitry with software programming techniques for providing project solutions is called as embedded system.

Purpose

By using this embedded system technology can be reduced the complexity of the circuits which further reduces to a great extent the cost and size

Design



Application

Embedded systems find numerous application in various field such as digital, electronics Telecommunications, computing

Networks, smart cards, satellite systems military defense system equipment, research equipment & so on ——

Q2 / part(b)

Device Driver

Devices are programs that translate general Operating system commands into specific references to hardware details,

Printers manufacturers, for Example create different MS-Windows device drivers for each model they sell.

often these devices drivers contain significant amount of assembly language code.

Q2 / Part(c)

Virtual Machine concept.

An effective way to explain how a Computer's hardware and software are related is called the virtual machine concept.

(PTO)

Programming Language Analogy :

↳ Each computer has a native machine language ~~is actually constructed about that~~ (Language L0) that runs directly on its hardware

↳ A more human friendly language is usually constructed above machine language called Language L1

Program written in L1 can run two different ways.

* Interpretation : L0 program interprets & executes L1 instruction one by one.

* Translation : L1 program is completely translated into an L0 program which then runs on the computer Hardware.

Q2 part D

Instruction Execution cycle :-

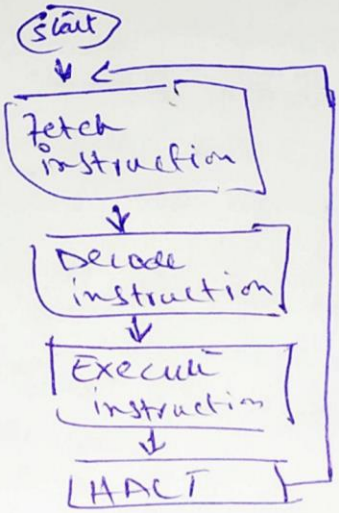
↳ the time period during which one instruction is fetched from & executed when computer given an instruction in machine language

11533

Pg 08

c) Each instruction is further divided into sequence of phases.

d) After Execution of program counter is incremented to point the next instruction.



Q2 part e

Motherboard chipset

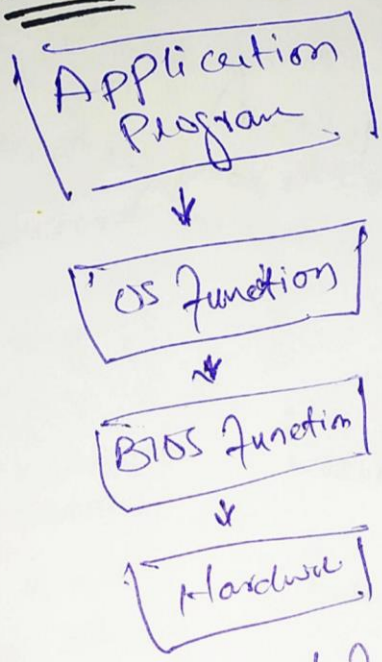
Living on the motherboard, PC's chipset controls the communication between the CPU, RAM, storage & other peripherals. The chipset determines how many high speed components of USB devices your motherboard can support.

PC chipset are designed by Intel & AMD but are found on motherboards from a variety of third party vendors such as MSI, Asus & ASRock.

11533

Q2 part(7)

Access levels for Input-Output operation



level 3: Call library functions to perform generic text I/O and file based I/O

level 2: Call the operating system function. If the OS uses GUI it has function to display graphics in a device independent way.

level 1: Call BIOS function to control colour, graphics, sound, keyboard input & low level disk I/O.

level 0: Send & receive data from H/W ports

Q2 part(g)

Basic part of Assembly language Instruction:-

1) Label (optional)
2) ~~Mnemonic~~ Mnemonic (required)
3) Operand (depends on the instruction)
4) Comment (optional)

- Data label
- code label
- constant expression
- constant
- Memory (data label)
- register

11533

Question No 03
Differentiate each of the following

Part a

Assembly language

- Program written for one processor will not run on another type of processor
- Performance & accuracy of Assembly language are better than high level.
- Directly read pointers at a physical address
- Assembler is used
- Should know about registers

High level language

- Program runs independently of processor type
- It is not as accurate as assembly language.
- Not possible to read pointers at a physical address
- Compiler is used
- Doesn't need detail about hardware like registers

Q3 part b

REAL MODE

1) Only 1MByte of memory can be addressed from hexadecimal 0000 to FFFF
2) The Processor can only run one program a time
3) Application programs are permitted to access any memory location including address that are directly link to system H/W
   Eg. MS DOS

PROTECTED MODE

1) It assigns each process a what of 4GByte of memory
2) Processor can run multiple program at the same time
3) Each program can be assigned its own reserved memory area and program are prevented accedining accessing each other's code & data
   eg MS window, Linux.

Q3 part(c)

Assembler

↳ Translates assembly language program into machine code (the output of assembler is called an object

Linker

↳ Computer program that links & merges various object files together in order to make executable file

Q3 part(d)

Instruction

↳ It's a task to be carried out by the processor at run time
Instruction are assembled into machine code & eventually linked into the final executable

Directives

↳ Its an instruction to the assembler telling it how to treat the data it's asked to assemble. for instance
↳ It might specify the location in memory where you want the program to be loaded. once it's been assembled.

Q3 part(e)

Code Label

These are just the names which are used by other instructions to jumps from particular position to that position where data label is called. It must End with (:)
Code labels are declared in coding of segment.

eg target
mov ax,bx
jmp target

Data label

These are just like variable is any other language

eg Num DWOR 1000
Data labels are declared in data segment of the code.

Q3 part(f)

Line Comment

Single line comments, beganning with a semicolon character (;) All the character following semicolon on the same line is ignored by assembler

Block Comment

Beganning with COMMENT Directive and user-specified symbol. All the subsequent lines of text are ignored by assembler until the same user specified symbol appears
COMMENT!

!

Q3 part g

Equal sign Directive

- Syntax
  name = expression
  ↳ expression is 32 bit integer (expression / constant)
  ↳ may be redefined.
  ↳ name is called symbolic constant
- good programming style
  ↳ use symbols

```
COUNT = 50
.
.
mov al, count
```

EQU Directive

- Define a symbol either an integer or text expression
- can't redefined.
- Syntax
  name EQU expression
  name EQU symbol
  name EQU <text>

eg.

```
matrix EQU 10 * 10
PI EQU <3.1416>
```

11533

Q4 Give the Answer to Each of the following

Part a
Concept of Portability

A language whose source program can be compiled and run on a wide variety of computer system is said to be portable.

Q4 Part(b)
Why would a high ---------- prefer?

A high level language may not be provided for direct hardware access. Even it does be awkward coding technique must often be used, resulting in possible maintain problem.

Q4 part(c)
Why does unicode invented?

Unicode is a universal computing standard to represent text in most writing system. It was invented to store most of the words character. It is stated during 1987.

(Q4 part (d))

$W = 11101100$, $X = 00010011$

$Y = 00111100$

$Z = W \vee X \wedge \neg Y$

$Y = 00111100$

$\neg Y = 11000011$

$X \wedge \neg Y =$

$$\begin{array}{r} 00010011 \\ 11000011 \\ \hline 00000011 \end{array} \quad \text{AND}$$

$W \vee (X \wedge \neg Y)$

$$\begin{array}{r} 11101100 \\ 00000011 \\ \hline 11101111 \end{array} \quad \text{OR}$$

Ans

(Q4 Part(e))

Circuit circuit table $\neg(A \vee B)$

A	B	A ∨ B	¬(A ∨ B)
F	F	F	T
F	T	T	F
T	F	T	F
T	T	T	F

11533

Q4 part (f)

Why does memory ---- ?

Convential memory is outside the CPU and it responds more slowly to access the request. Registers are hard wired inside the CPU.

Q4 part (g)

Discuss the basic ---- ?

XMM Registers

The x86 architecture also contain eight 128-bit registes called streaming SIMD extension to the instruction

MMX Register

MMX technology improves the performance of intel processor when implementing advanced multimedia and communication application

The 64 bit MMX register support special instruction called SIMD (single instruction Multiple data.

11533

Segment Register

In real address mode 16 bit segment register indicate base addresses of preassigned memory areas named segment.

Basic Programming Execution Registers

Registers are high speed storage location inside the CPU designed to be accessed at much higher speed than conventional memory when a processing unit is optimized for speed.

11533

Q5. Discuss the following.

INCLUDE

Insert source code from the source file given filename into the current source file during assembly.

Syntax

INCLUDE filename

.386

Enables assembly of nonprivileged instructions for the 80386 processor. disables assembly of instructions introduced with later processors.

.MODEL

Initialize the program memory model. The memory model can be, TINY, SMALL, COMPACT MEDIUM, LARGE, HUGE or FLAT

the langtype can be, C, BASIC, FORTRN.

.STACK

When used with .Model defines a stack segment. the optional size specifies the number of bytes for the stack

.PROTO

~~Label PROTO~~ The PROTO directive by using the INVOKE directive.

Syntax

label PROTO [distance] [language type]

[[parameter]: tag]]

- .DATA

When used with model, starts a near data segment for initialized data

- .CODE

When used with .MODEL indicates the start of a code segment called.

- PROC

Set of code that can be branched to and returned from in such a way.

- ENDP

Mark the End of the Procedure name previously begin with PROC

- END

Mark the end of modul and optionally sets the program entry point to a address.

11533

Question No 6

Part a

Write a program ---- A = (A+B) + (C+D)

```
.386
.model flat, stdcall
.stack 4096
ExitProcess PROTO
dwExitCode:DWORD
.code
main PROC
    mov eax, 3h
    mov ebx, 8h
    mov ecx, 1h
    mov edx, 8h
    add eax, ebx
    add ecx, edx
    Sub eax, ecx
    INVOKE ExitProcess, 0
main ENDP
END main.
```

Q6 part(b) show that ---------- 12345678h

0000	78
0001	56
0002	34
0003	12

Q6 Part (c)
write a statement ---- ,0

```
.data
String1 byte "Assembly language is easy",0
stringsize byte ?
.code
mov eax, SIZEOF string1
mov stringsize, eax.
```

Q6 part (d)
write a program ----

Let the Arithematic operation is

$$x = -m + (n - o)$$

```
.data
xDWORD?        ; uninitialized variable
m DWORD 12     ; initialized variable m
nDWORD 25      ; initialized variable n
o DWORD 17     ; initialized variable o

.code
main PROC
mov eax, m     ; moves value of m into eax
neg eax        ; EAX = -12
mov ebx, n     ; mov value of n into ebx
sub ebx, o     ; Ebx = 13
add eax, ebx   ; -12 + 13
mov x, eax     ; (1)
Call DumpRegs
exit
main ENDP
END Main
```