UNISA university of south africa

COS2621

MAY/JUNE 2018

COMPUTER ORGANIZATION

STUDENT NUMBER									

IDENTITY NUMBER												

FOR USE BY EXAMINATION INVIGILATOR

Question No	Marks					
	Examiners					
	1		2		3	
Q1						
Q2						
Q3						
Total						

Subject

Number of paper

Date of examination

Examination centre

WARNING

1 A candidate who without authorisation takes into the examination venue any book document or object which could assist him in the examination and does not hand over such material to the invigilator before the official commencement of the examination will be guilty of infringing the University's examination regulations and will be liable to punishment as determined by Council

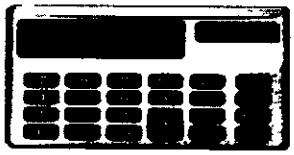
2 Rough work may be done only on the examination question paper and must be labelled as such

3 No notes may be made on any part of the body such as the hands or on any garment

4 This page/paper is the property of the University and under no circumstances may the candidate retain it or take it out of the examination venue

NB PLEASE COMPLETE THE ATTENDANCE REGISTER ON THE BACK PAGE, TEAR OFF AND HAND TO THE INVIGILATOR

UNIVERSITY EXAMINATIONS



UNIVERSITEITSEKSAMENS

UNISA university of south africa

COS2621

May/June 2018

COMPUTER ORGANISATION

Duration 2 Hours

100 Marks

EXAMINERS :

FIRST : MR W FRIEDRICH
SECOND : MRS S VALLABHAPURAPU

Use of a non-programmable pocket calculator is permissible.

Closed book examination

This paper consists of 12 pages plus an appendix.

INSTRUCTIONS

1 Answer question 1 on the answer sheet provided on p. 2 of the examination paper and questions 2, and 3 in the space provided below the question. Please do not detach any sheets from the question and answer book.

2. All rough work must be done in this question and answer book

3. The mark for each question is given in brackets next to the question.

4. The appendix contains a list of x86 Assembly Language instructions

ALL THE BEST!

Question 1 (Answer Sheet) [30]

NB Any answers from question 1 not answered on this answer sheet will not be marked. Each question will count 2 marks.

Question 1	Write down the alternative you choose (1 - 5)
(1.1)	
(1.2)	
(1.3)	
(1.4)	
(1.5)	
(1.6)	
(1.7)	
(1.8)	
(1.9)	
(1.10)	
(1 11)	
(1.12)	
(1.13)	
(1.14)	
(1.15)	

Question 1 **[30]**

Indicate which one of alternatives (1) to (5) you consider to be the correct one by writing down your answer on the answer sheet provided on the previous page Each question counts 2 marks.

1 1 Which one of the following statements is false when we are talking about ASCII encodings?

(1) ASCII is a fixed length code.
(2) The last 4 bits for each digit (0-9) are identical to the binary representation of the digit.
(3) When using 8 bits to store characters, the rightmost bit is always set to zero.
(4) The ASCII representation of a space is 20h.
(5) When one wants to convert an ASCII character ('0' to '9') to its numerical value, we can simply subtract 30h from the ASCII encoding

1 2 Which one of the following binary codes have even parity (assume that the left-most bit is the parity bit)?

(1) 10101110
(2) 10101010
(3) 10100111
(4) 11110001
(5) 00000001

1 3 Calculate $(3A69)_{16} + (B2E1)_{16}$

(1) $(ED4A)_{16}$
(2) $(AD50)_{16}$
(3) $(FD5A)_{16}$
(4) $(FE4A)_{16}$
(5) None of the answers given above is correct.

ROUGH WORK

1 4 Calculate $(74)_{16}$ - $(3A)_{16}$

(1) $4A_{16}$
(2) $(10)_{10}$
(3) $(10)_{16}$
(4) $(49)_{16}$
(5) $3A_{16}$

1 5 Which one of the following is a representation of an AND gate?

(1) z'y'
(2) (x + y)'
(3) (x'y')'
(4) (x' + y')'
(5) none of the above

1 6 _____ provide storage internal to the CPU.

(1) Control units
(2) ALUs
(3) Main memory
(4) Registers
(5) 4 and 3

1 7 Give the result of the following binary operation
A XOR B, where A = *1010 1101* and B = *0011 1001*

(1) 1011 1101
(2) 0010 1001
(3) 0000 0000
(4) 0101 0010
(5) None of the above

ROUGH WORK

1 8 You have to design a binary gate (i.e. two inputs) with the output equal to 1 if both inputs are equal to 0 or both equal to 1. If this is not the case, the output is equal to 0. Which one of the following Boolean functions is a representation of such a gate?

(1) x'y' + xy
(2) x'y + xy'
(3) x'y' + x'y + xy' + xy
(4) x'y' + x'y'
(5) none of the above

1 9 Suppose a machine has fifteen (15) general-purpose 32-bit registers How many bits must be reserved in the machine code instruction in order to address any one of these fifteen registers?

(1) 1
(2) 2
(3) 3
(4) 4
(5) 5

1 10 Which one of the following instruction would you use if you want to branch to the label TEST if the sign flag is set?

(1) JN TEST
(2) JZ TEST
(3) JS TEST
(4) JNS TEST
(5) JNP TEST

ROUGH WORK

1 11 If we subtract AX from BX and we want to branch to the label `next` if the result is negative, we will use the instruction _____.

(1) jng next
(2) jnc next
(3) jne next
(4) jn next
(5) None of the above

1 12 If we want to read a string of characters from the keyboard using an `int 21h` instruction, the AH register should be set to ____.

(1) 01h
(2) 02h
(3) 07h
(4) 0Ah
(5) 41h

1.13 Suppose AL contains the value 8Ah. What will be the result of the following instruction?

```
sar al,3
```

(1) 11h
(2) E2h
(3) 50h
(4) F1h
(5) None of the above

ROUGH WORK

1 14 Where is the result of the following operation stored?

```
imul cl
```

(1) AX
(2) CX
(3) AX:DX
(4) DX:AX

(5) None of the above

1.15 Give the postfix representation of the following expression where infix notation is used.

(A + B * C) – (E / F)

1. AB+C*EF/–
2. ABC*+EF/–
3. AB+C*EF–/
4 ABC+*EF–/
5. None of the above

ROUGH WORK

Question 2

[40]

Write your answers down in the space provided below the question.

2.1 How long will it take to execute the following code using the assumptions given below? (5)

```
mov  ax,2h
mov  bx,3h
mul  bx
```

```
add  ax,bx
```

Assume the following

(i) The machine on which the above program fragment is to be executed has a 800 MHz clock
(ii) Four cycles are needed to fetch an instruction from memory (Instructions are not prefetched.)
(iii) Multiplication takes 5 clock cycles to execute
(iv) Addition takes 4 cycles to execute
(v) The mov instruction takes 2 cycles to execute
(vi) Fetch and execute cycles do not overlap

Express your answer in microseconds (μsec), where 1 μsec = 1×10^{-6} sec

Note A machine speed of 800 MHZ means that the machine completes 800×10^6 cycles per second This means that one cycle takes $1/(800 \times 10^6)$ seconds to complete, i.e $1/800 \times 10^{-6}$ sec = 0.00125 μsec.

2.2 Discuss the disadvantages of using DEBUG to develop assembly language programs (8)

2 3 Explain the four Intel x86 CPU General Purpose registers and its components, also provide a diagram to support your explanation. (15)

2.4 What relevance would the study and/or use of assembly languages have in the future? (5)

2.5 Simplify ((B'+C')' • (A'B))' + A using Boolean identities. Show all your steps, (4)

2 6 The following table reflects the memory contents of a part of memory in a one-address machine with an accumulator

Address	Contents
300	320
305	310
310	310
315	325
320	305
325	300

What values do the following instructions load into the accumulator?

LOAD IMMEDIATE 310
LOAD DIRECT 305
LOAD INDIRECT 325

(3)

Question 3 [30]

3.1 Give examples of x86 assembler instructions in which the following addressing modes are used:

1. Immediate addressing
2. Direct addressing
3. Indexed addressing
4. Stack addressing
5. Base addressing (5)

3.2 Give the x86 assembler code to implement the following program.

Consider the following state reached during a DEBUG session:

```
AX=8A6C  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0C61  ES=0C61  SS=0C61  CS=0C61  IP=010A   OV UP EI NG NZ NA PE CY
0C61:010A 8B04          MOV     AX,[SI]                         DS:0000=20CD
-
```

(a) What is the effective 20-bit address of the next instruction that will be executed? (2)

(b) Give the hexadecimal representation of the next instruction that will be executed. (1)

(c) What addressing mode is used in the next instruction to be executed? (1)

(d) What value that will be moved to AX when the next instruction is executed? (1)

(e) What is the status of the *overflow flag* and the *sign flag* respectively at the current state? Refer to the DEBUG display given above to motivate your answer (2)

(f) How many bytes does the next instruction that will be executed occupy? (1)

(g) If the previous instruction was a 16-bit integer multiplication instruction (IMUL), what is the value of the quotient and remainder respectively? (2)

3.3 We can use various methods to double the value n, stored in the AX register Three possibilities are n + n, 2 * n and n - (- n). Write sequences of assembly language instructions that doubles the contents of AX using each of the three methods described above (15)

APPENDIX

DOS INTERRUPT 20h - terminate program		
DOS INTERRUPT 21h		
AH	**Purpose**	**Description**
0	Terminate program	Terminates execution of program
1	Keyboard input	Waits for keyboard input, displays it and returns it in AL
2	Display output	Displays the character in DL on the screen
3	Auxiliary input	Waits for a character from the communications port and puts it in AL
4	Auxiliary output	Outputs the character in DL to the communications port
5	Printer output	Outputs the character in DL to the printer
6	Keyboard input Screen output	Performs both input and output It can also determine the input status Unlike the other input functions, it does not wait for an input character Also note that the input character is not automatically displayed and that the Ctrl-Break command does not terminate the operation As usual, the DL register contains the output character and the AL register receives the input character You ask for input by placing the value FFh in the DL register On return from the function, the zero flag is set if no character is ready If the zero flag is clear, it means a character was read into the AL register If the DL register contains any value other than FFh, that character is sent to the standard output device Also compare with functions 7 and 8
7	Keyboard input no echo	Waits for a character from the keyboard and puts it in AL The character is not displayed on the screen
8	Keyboard input no echo	Waits for a character from the keyboard, returns it in AL and terminates when Ctrl-Break is pressed
9	Display string	We have seen that functions 2 and 6 can display single characters, but function 9 is much easier to use for more than one character Of course, non-displayable characters (such as carriage return, line feed, and Esc) can also be included in the string To use this function, you place the string somewhere in memory and terminate it with a dollar sign The address of the string is placed in the DS DX registers Set AH register to 9 and execute the INT 21h instruction
0A	Buffered keyboard	Reads characters from the keyboard into memory buffer DS DX points to the input buffer 1st byte = max number of characters and 2nd byte = actual number of characters entered The string is stored from the third byte onwards
0B	Check keyboard	Checks to see if a character is available from the keyboard Sets AL=FFh if a Status character is ready Otherwise AL=0
0C	Clear keyboard	Clears keyboard buffer and performs the INT 21h keyboard input function
4C	Terminate program	Returns control to DOS

INSTRUCTIONS

(Note that the first operand is always the destination operand)

ADD: **ADD** binary numbers Adds the source operand to the destination operand and stores the result in the destination operand
Format: ADD operand1,operand2 **Flags:** OF, SF, ZF, AF, PF, CF

AND: **ANDs** the bits in the source and destination operands The result is stored in the destination
Format: AND operand1,operand2 **Flags:** OF, SF, ZF, AF=?, PF, CF=0

CALL: **CALLs** a near or far procedure Decrements the SP (stack pointer) by 2 (or 4 for a far call)
If the procedure is **NEAR** (in the same segment) Pushes the current location (only the offset) of the next instruction onto the stack and transfers control to the destination If the procedure is **FAR** (in a different segment) Pushes the current location (both segment and offset) of the next instruction onto the stack and transfers control to the destination
Format: CALL operand **Flags:** None

CLC: CLear Carry Flag
Format: CLC **Flags:** CF=0

CMP: CoMPares the contents of operand1 and operand2. Compares the destination to the source by doing an implied subtraction of the source from the destination The values of the operands do not change
Format: CMP operand1,operand2 **Flags:** OF, SF, ZF, AF, PF, CF

DB/DW/DD: These operations are instructions to the assembler
DB Allocates one byte of storage
DW Allocates two bytes of storage
DD Allocates four bytes of storage

DEC: DECrement. Subtracts 1 from a byte or word
Format: DEC reg **Flags:** OF, SF, ZF, AF, PF
DEC mem

DIV: DIVide, unsigned. Divides an unsigned (the leftmost bit is treated as part of the data and not as a sign) dividend by an unsigned divisor. If the divisor is 8 bits, the dividend is assumed to be in AX, the quotient is stored in AL and the remainder in AH If the divisor is 16 bits, the dividend is assumed to be in DX AX, the quotient is stored in AX and the remainder in DX.
Format: DIV reg **Flags:** all undefined
DIV mem

HLT: HaLT. Stops the CPU until a hardware interrupt occurs.
Format: HLT **Flags:** None

IDIV: Integer (signed) DIVision. Divides a signed dividend by a signed divisor. Compare with DIV.

Dividend (operand1)	**Divisor** (operand2)	**Quotient**	**Remainder**
AX (16 bits)	reg/mem (8 bits)	AL	AH
DX:AX (32 bits)	reg/mem (16 bits)	AX	DX

Format: IDIV reg **Flags:** AF, CF, OF, PF, SF, ZF (all undefined)
IDIV mem
(We need only specify operand2 as either AX or AL is used as operand1 by default.)

IMUL: Integer (signed) MULtiplication Performs a signed integer multiplication on either AL or AX Compare with MUL.

Multiplicand (operand1)	**Multiplier** (operand2)	**Product**
AL (8 bits)	reg/mem (8 bits)	AX
AX (16 bits)	reg/mem(16 bits)	DX:AX

Format: IMUL reg **Flags:** OF, CF, SF, ZF, AF and PF (all undefined)
IMUL mem
(We need only specify operand2 as either AX or AL is used as operand1 by default.)

INC: INCrement. Adds 1 to a register or memory position The operation **does not affect the carry flag.**
Format: INC reg **Flags:** OF, SF, ZF, AF, PF
INC mem

INT: INTerrupt. Generates a software interrupt, which in turn calls a routine in BIOS or DOS
Format: INT immed **Flags:** TF, IF

JMP: Unconditional JuMP. Jumps unconditionally to a label or memory position
Format: JMP operand **Flags:** None

Conditional Jumps: Jcondition. Jumps if a specific flag condition is true.
JA/JNBE Jump if Above or Jump if Not Below/Equal
JAE/JNB Jump if Above/Equal or Jump if Not Below
JB/JNAE Jump if Below or Jump if Not Above/Equal.
JBE/JNA Jump if Below/Equal or Jump if Not Above.

JC Jump if Carry
JCXZ/JECXZ Jump if CX is Zero or if ECX is Zero
JE/JZ Jump if Equal or Jump if Zero
JG/JNLE Jump if Greater or Jump if Not Less/Equal
JGE/JNL Jump if Greater/Equal or Jump if Not Less
JL/JNGE Jump if Less or Jump if Not Greater/Equal
JLE/JNG Jump if Less/Equal or Jump if Not Greater
JNC Jump if No Carry
JNE/JNZ Jump if Not Equal or Jump if Not Zero
JNO Jump if No Overflow
JNP/JPO Jump if No Parity or Jump if Parity Odd
JNS Jump if No Sign - jumps if an operation set the sign to positive
JO Jump if an operation caused an Overflow
JP/JPE Jump on Parity or Jump if Parity Even
JS Jump if an operation set the Sign to negative
Format: Jump_condition short_label **Flags:** None in all cases
Jump_condition mem

LEA: Load Effective Address Calculates and loads the 16-bit effective address (offset) of a memory position into a register
Format: LEA reg,mem **Flags:** None

LOOP: **LOOP** until complete Executes a routine a specified number of times The CX register must be loaded with the iteration count before the start of the loop **Loop** appears at the end of the loop. it decrements CX and causes a jump if CX is not equal to zero. If CX=0, the next instruction after LOOP is executed
Format: LOOP short_label **Flags:** None

LOOPE/LOOPZ: **LOOP** while Equal or while Zero These instructions are similar to **LOOP** except that they terminate if CX=0 **or** if the ZF flag is set to zero (i e a non-zero condition)

LOOPNE/ **LOOP** while Not Equal or loop while Not Zero Similar to **LOOP** except that they
LOOPNZ: terminate if CX=0 or if the ZF flag is set to 1 (zero condition)

MOV: **MOV**e data Moves (actually copies) a byte or word from a source (operand2) to a destination operand (operand1)
Format: MOV operand1, operand2 **Flags:** None

MUL: **MUL**tiply unsigned integers Multiplies AX or AL by a source operand of 16 or 8 bytes respectively In the first case the product is stored in AX DX and in the second case the product is stored in AX

	Multiplicand (operand1)	**Multiplier** (operand2)	**Product**
	AX (16 bits)	reg/mem (16 bits)	DX AX
	AL (8 bits)	reg/mem (8 bits)	AX

Format: MUL reg **Flags:** Affects CF and OF AF, PF, SF and ZF are
MUL mem undefined
(We need only specify operand2 as either AX or AL is used as operand1 by default)

NEG: **NEG**ate. NEG calculates the two's complement of a binary value
Format: NEG operand **Flags:** OF, SF, ZF, AF, PF, CF

NOP. **NO OP**eration. Do nothing.
Format: NOP **Flags:** None

NOT: Performs a logical **NOT** on an operand
Format: NOT operand **Flags:** None

OR: Performs a logical OR on two operands The result is stored in operand 1 (the destination operand).
Format: OR operand1,operand2 **Flags:** OF=0, CF=0, SF, ZF, and PF are affected AF is undefined

POP: **POP**s a word from the stack into the operand
Format: POP operand **Flags:** None

PUSH: **PUSH**es a word onto the stack
Format: PUSH operand **Flags:** None

RCL/RCR: **R**otate bits **L**eft or **R**ight through the **C**arry flag
RCL rotates (shifts) the destination operand to the left a number of times as specified by the source operand. The carry flag is copied into the least significant bit and the most significant bit is copied into the carry flag with each shift The CL register or a constant may be used to control the number of rotations
RCR rotates (shifts) the destination operand to the right a number of times as specified by the source operand The carry flag is copied into the most significant bit and the least significant bit is copied into the carry flag with each shift The CL register or a constant may be used to control the number of rotations
Format: RCL/RCR reg,1 RCL/RCR mem,1 **Flags:** OF, CF
RCL/RCR reg,CL RCR/RCR mem,CL

REP/REPE/REPZ/ REPeat string Used as a prefix to
MOVS/MOVSB/MOVSW/MOVSD/REPNE/REPNZ operations

REP repeats a string operation a specified number of times CX is used as a counter and is decremented each time the instruction is repeated The operation repeats until CX=0
REPE/REPZ repeats a string operation until CX=0 or ZF=0
REPNZ/REPNE repeats a string operation until CX=0 or ZF=1
Format: REP/REPE/REPZ/REPNE/REPNZ string_operation **Flags:** Depends on the associated string operation

RET/RETN/RETF. RETurn from a procedure previously entered by a **NEAR** or **FAR** CALL.
NEAR **RET/RETN** (return near) pops the word at the top of the stack into IP and increments SP by 2
FAR **RET/RETF** pops the 2 words at the top of the stack into IP and CS and increments SP by 4
An optional immediate 8_bit operand (called a pop_value) is added to the SP
Format: RET **Flags:** None
RET operand

ROL/ROR: Rotate Left or **R**ight
ROL rotates (shifts) the destination operand to the left a number of times as specified by the source operand The leftmost (most significant) bit is rotated into the least significant position The most significant bit also moves into the carry flag for each shift. The CL register or a constant may be used to control the number of rotations
ROR rotates (shifts) the destination operand to the right a number of times as specified by the source operand The rightmost (least significant) bit rotates into the most significant position The least significant bit also moves to the carry flag for each shift The CL register or a constant may be used to control the number of rotations
Format: ROL/ROR reg,1 ROL/ROR mem,1 **Flags:** OF,CF
ROL/ROR reg,CL ROL/ROR mem,CL

SAL/SAR: **S**hift **A**rithmetic **L**eft or **R**ight
SAL shifts the destination operand to the left a number of times as specified by the source operand. Zero is shifted into the least significant bit and the most significant bit is shifted into the carry flag for each shift The CL register or a constant may be used as a source operand.
SAR shifts the destination operand to the right a number of times as specified by the source operand. The most significant bit retains its previous value which means that the sign bit is duplicated with each shift. The least significant bit is shifted into the carry flag with each shift. The CL register or a constant may be used as source operand
Format: SAL/SAR reg,1 **Flags:** CF,OF,PF,SF,ZF (AF undefined)
SAL/SAR mem,1
SAL/SAR reg,CL
SAL/SAR mem,CL

SBB: **S**u**B**tract with **B**orrow Subtracts the source operand from the destination operand and then subtracts the value of CF from the destination SBB is used in multiword subtraction to carry an overflowed 1 bit into the next stage of arithmetic

Format: SBB reg,reg SBB reg,immed
SBB reg,mem SBB mem,immed
SBB mem,reg SBB AL,immed

SBB AX,immed

Flags: OF, SF, ZF, AF, PF, CF

SCAS/SCASB/SCASW: **SCA**n String Scans a string in memory pointed to by ES DI for a value that matches a value in AL or AX
Format: SCAS operand SCASB (match with AL)
SCAS ES operand SCASW (match with AX)

Flags: OF, SF, ZF, AF, PF, CF

SHL/SHR: **SH**ift logical **L**eft or **SH**ift logical **R**ight
SHL is identical to SAL.
SHR shifts bits to the right a specified number of times and fills the bits on the left with 0's
The CL register or a constant (for some assemblers the constant may only be a 1) may be used to control the number of shifts
Formats: SHL/SHR reg,immed SHL/SHR reg,CL
SHL/SHR mem,immed SHL/SHR mem,CL

Flags: OF, SF, ZF, AF (undefined), PF, CF

STC: **S**e**T** **C**arry flag. Sets the CF to 1
Format: STC **Flags:** CF=1

STD: **S**e**T** **D**irection flag Sets the DF flag to 1 This causes SI and/or DI to be decremented by string operations which in turn causes string operations to process strings from right to left
Format: STD **Flags:** DF=1

STOS/STOSB/ Store string
STOSW/STOSD Stores either AL (byte) or AX (word) in the memory position addressed by ES DI DI is incremented if DF=0 and DI is decremented if DF=1 (compare with STD)
Format: STOS mem STOSB (stores AL) **Flags:** None
STOS ES mem STOSW (stores AX)

SUB: **SUB**tract Subtracts the source operand from the destination operand

Format: SUB operand1,operand2 **Flags:** OF, SF, ZF, AF, PF, CF

TEST: **TEST** bits Tests individual bits in the destination operand against those in the source operand. TEST performs a logical AND, but the **destination operand is not affected**
Format: TEST reg,reg TEST reg,immed
TEST reg,mem TEST mem,immed
TEST mem,reg TEST AL,immed
TEST AX,immed
Flags: OF=0, SF, ZF, AF (undefined), PF, CF=0

XCHG: **EXCH**an**G**es operands Exchanges the contents of the two operands
Format: XCHG operand1,operand2 **Flags** None

XOR: **EX**clusive **OR**. Performs a logical exclusive OR on the bits in two operands
Format: XOR operand1,operand2 **Flags:** OF=0, SF, ZF, AF (undefined), PF, CF=0

ASCII TABLE

ASCII-code	Character	ASCII-code	Character	ASCII-code	Character
0000 0000	NUL	0011 0000	0	0110 0000	
0000 0001	SOH	0011 0001	1	0110 0001	a
0000 0010	STX	0011 0010	2	0110 0010	b
0000 0011	ETX	0011 0011	3	0110 0011	c
0000 0100	EOT	0011 0100	4	0110 0100	d
0000 0101	ENQ	0011 0101	5	0110 0101	e
0000 0110	ACK	0011 0110	6	0110 0110	f
0000 0111	BEL	0011 0111	7	0110 0111	g
0000 1000	BS	0011 1000	8	0110 1000	h
0000 1001	HT Tab	0011 1001	9	0110 1001	i
0000 1010	LF	0011 1010		0110 1010	j
0000 1011	VT	0011 1011	,	0110 1011	k
0000 1100	FF	0011 1100	<	0110 1100	l
0000 1101	CR	0011 1101	=	0110 1101	m
0000 1110	SO	0011 1110	>	0110 1110	n
0000 1111	SI	0011 1111	?	0110 1111	o
0001 0000	DLE	0100 0000	@	0111 0000	p
0001 0001	DC1	0100 0001	A	0111 0001	q
0001 0010	DC2	0100 0010	B	0111 0010	r
0001 0011	DC3	0100 0011	C	0111 0011	s
0001 0100	DC4	0100 0100	D	0111 0100	t
0001 0101	NAK	0100 0101	E	0111 0101	u
0001 0110	SYN	0100 0110	F	0111 0110	v
0001 0111	ETB	0100 0111	G	0111 0111	w
0001 1000	CAN	0100 1000	H	0111 1000	x
0001 1001	EM	0100 1001	I	0111 1001	y
0001 1010	SUB	0100 1010	J	0111 1010	z
0001 1011	ESC	0100 1011	K	0111 1011	{
0001 1100	FS	0100 1100	L	0111 1100	|
0001 1101	GS	0100 1101	M	0111 1101	}
0001 1110	RS	0100 1110	N	0111 1110	~
0001 1111	US	0100 1111	O	0111 1111	Delete
0010 0000	space	0101 0000	P		
0010 0001	!	0101 0001	Q		
0010 0010	"	0101 0010	R		
0010 0011	#	0101 0011	S		
0010 0100	$	0101 0100	T		
0010 0101	%	0101 0101	U		
0010 0110	&	0101 0110	V		
0010 0111	'	0101 0111	W		
0010 1000	(	0101 1000	X		
0010 1001	)	0101 1001	Y		
0010 1010	*	0101 1010	Z		
0010 1011	+	0101 1011	[		
0010 1100	,	0101 1100	\		
0010 1101	-	0101 1101	]		
0010 1110		0101 1110	^		
0010 1111	/	0101 1111	_		

Tear

attendance register (university copy) UNISA

Fill-in/MCQ

Examination period

Student number

Surname

First Names

Subject

Code of paper — Number of paper

Centre Date

This is to certify that I have read the rules governing the examinations as set out on the inside cover of this examination answer book and in the examination instructions

That the information supplied by me in this answer book is correct and valid

I undertake to adhere to the procedures, rules and regulations of the University of South Africa as published in the official brochures

Signature of candidate

ID Number

Batch No
28092015MCQ

Signature of invigilator

UNISA invigilator's personnel number

NOTE Not a valid document if not completed by the Invigilator

Tear

attendance register (student copy) UNISA

Fill-in/MCQ

Examination period

Student number

Surname

First Names

Subject

Code of paper — Number of paper

Centre Date

This is to certify that I have read the rules governing the examinations as set out on the inside cover of this examination answer book and in the examination instructions

That the information supplied by me in this answer book is correct and valid

I undertake to adhere to the procedures, rules and regulations of the University of South Africa as published in the official brochures

Signature of candidate

ID Number

Batch No
28092015MCQ

Signature of invigilator

UNISA invigilator's personnel number

NOTE Not a valid document if not completed by the Invigilator