GX2649

OLLSCOIL NA hÉIREANN, GAILLIMH
THE NATIONAL UNIVERSITY OF IRELAND, GALWAY

AUTUMN EXAMINATIONS 1999

THIRD YEAR ELECTRONIC ENGINEERING EXAMINATION

COMPUTER SYSTEMS ENGINEERING I

Professor L. E. Davis
Professor D.J. Wilcox
Dr. E. Jones

Duration of Examination: **3 hours**

Instructions to candidates: Answer **FIVE** questions.
All questions carry equal marks.

1. (a) Explain what is meant by the terms "open-collector output" and "Tri-State output". Describe two application areas where gates with open-collector outputs are useful. What are the advantages and disadvantages of such gates ? Describe one application where gates with Tri-State outputs are useful.
[10 marks]

(b) For the circuit shown in Fig. 1 (overleaf), calculate the range in which the value of the pull-up resistor R_L should fall. The following values may be used for the relevant electrical characteristics of the logic gates: I_{oh} = -220 μA, I_{ih} = 45 μA, $V_{OH(min)}$ = 2.4 V, $V_{OL(max)}$ = 0.4 V, $I_{ol(max)}$ = 16 mA, I_{il} = -1.6 mA). You may assume that only one gate is "on" in the low output state.
[10 marks]

2. (a) Using Karnaugh mapping, find a minimal Sum of Products expression for the following function:

$$F(A, B, C, D, E) = \Sigma (5, 7, 13, 15, 16, 20, 25, 27, 29, 31).$$

[10 marks]

(b) Design a one-out-of-four checker with four inputs, A, B, C and D, and a single output, ERR. The output should be high if two or more inputs are high, and low if only one, or none, of the inputs is high.
[10 marks]

3. Design a 3-bit Odd/Even Up-Counter, that will count in the sequence 1, 3, 5, 7, 1, 3 etc. if the value of a MODE control input is high, and will count in the sequence 0, 2, 4, 6, 0, 2 etc. if this control input is low. Represent the system using using an ASM chart, and implement the system using your own choice of D or J-K flip-flops. Justify your choice of flip-flop type.
[20 marks]

[cont'd ...]

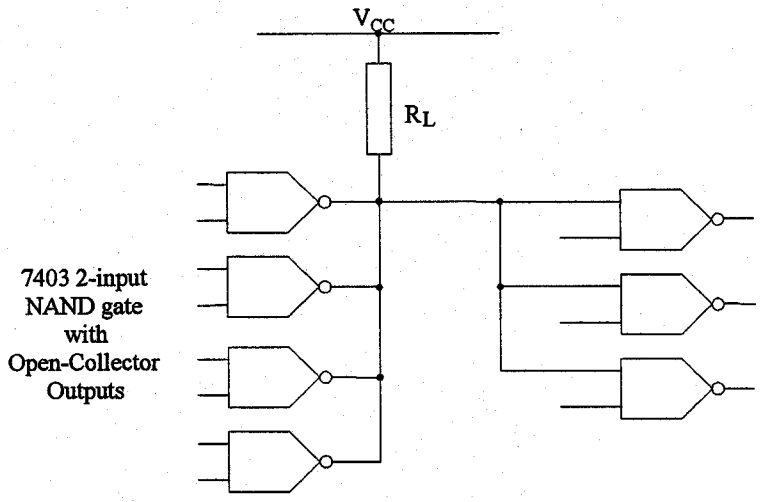

Fig. 1. Open-collector configuration (Q. 1)

4. (a) Explain how single-precision floating point numbers are stored according to IEEE Standard 754. Show how the following numbers would be stored according to this standard (single-precision):

   (i) $1001.001_2 \times 2^6$;

   (ii) -2113.625_{10}.

   [10 marks]

   (b) The source data word 1001 is to be transmitted using a $H_{7,4}$ Hamming code. Show how the check bits are calculated before transmission. If data bit D_2 is changed from a 0 to a 1 during transmission, show how the receiver carries out error detection and correction.

   [10 marks]

5. An 8086-based PC is used in an alarm system to monitor the SO_2 emissions from a chemical plant (see Fig. 5). The SO_2 detector outputs an analogue voltage which is proportional to the SO_2 concentration. An analogue to digital converter (ADC) converts this voltage (0 - 255 mV) into an 8-bit unsigned binary number. A separate timing circuit provides a start convert pulse (SC) to the ADC at regular intervals, and, on receipt of an end-of-conversion signal (EOC) from the ADC, enables the INTR input of the 8086. The Interrupt Service Routine (ISR) for this interrupt (vector number 50) reads the 8-bit number from Port A of the 8255, and, if the voltage is greater than or equal to 100 mV, sounds an alarm by outputting a square wave (frequency 5 kHz and duty cycle 50%) continually on PC_0. A human operator must then intervene to manually reset the computer and attend to the alarm. Write an Assembly language program and Interrupt Service Routine to accomplish this.

   The following points should be noted:
   (i) The ISR for the interrupt is stored at location 20000H in memory.
   (ii) The control word required to configure the 8255 is 90H; the base address of the 8255 is 0300H.

   [20 marks]

[cont'd ...]

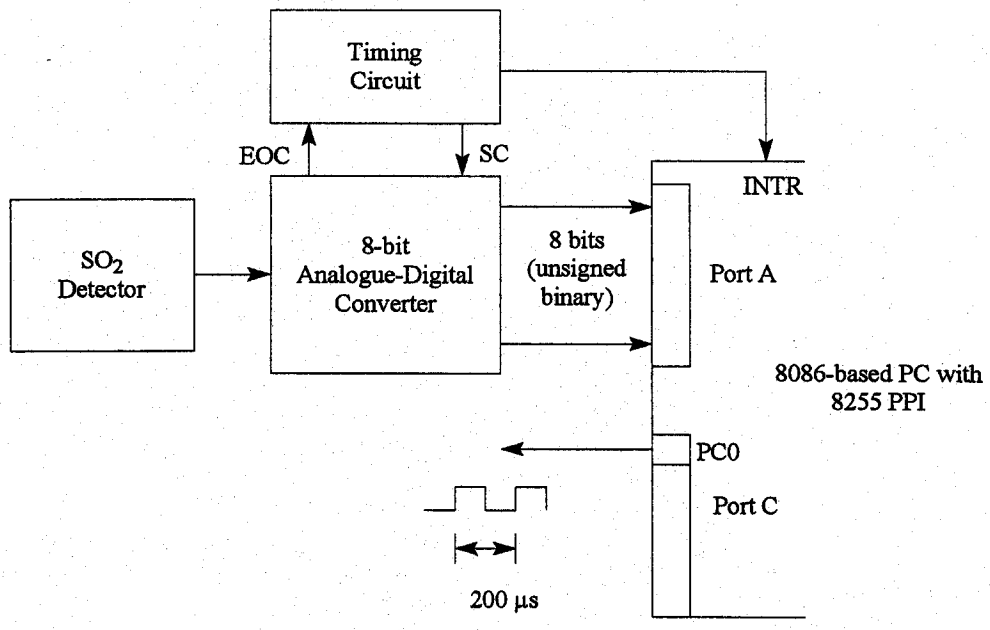

Fig. 5. Block diagram of SO_2 alarm system (Q.5).

6. An 8086-based PC equipped with an 8255 Programmable Peripheral Interface (PPI) is to be used to test the logical behaviour of 7400 TTL quad 2-input NAND gates on a production line. The configuration is shown in Fig. 2. Write an Assembly language program which will logically test all four gates for all input combinations. The program should print out a message (using INT 21H, function 9) indicating whether the chip is functioning correctly or not.

Note the following:

(i) For INT 21H, function 9, the offset of the address of the string to be printed on the screen must be placed in DX. The function number is placed in AH.

(ii) The control word required to configure the 8255 PPI is 82H, and the base address is 0300H.

(iii) Your program should be adequately commented.

[20 marks]

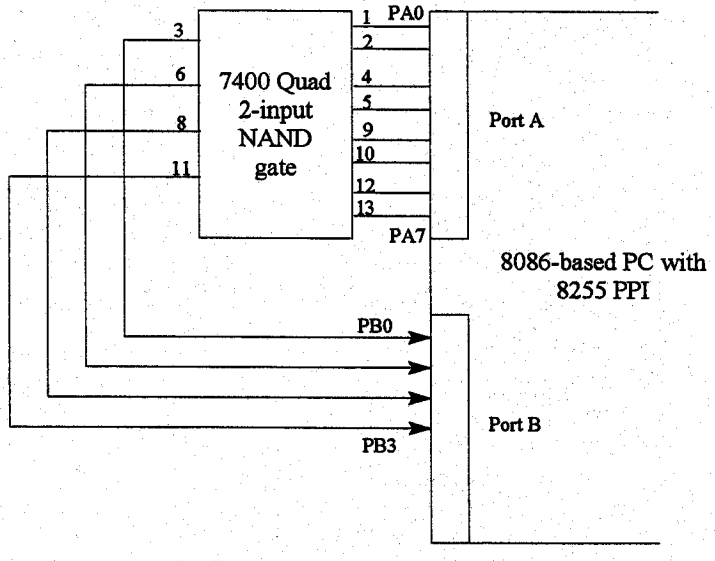

Fig. 6. Block diagram of TTL test configuration (Q.6)

7. A simple digital control system is to be implemented using an 8086-based embedded system that includes an 8255 PPI. The control system is required to monitor an externally-generated signal from a sensor, and filter it using a first-order digital filter with the following difference equation:

$$y(n) = 0.5x(n) + 0.25y(n-1)$$

The input signal x(n) is available in binary form from an Analogue to Digital converter (ADC) connected to Port A of the 8255, while the output of the low-pass filter should be sent to a Digital to Analogue converter (DAC) connected to Port B. Timing is controlled by means of an external clock generation circuit connected to Bit 0 of Port C; when a low-to-high transition is detected on this bit, the PC should read in the input sample, process it using the digital filter, then send the filtered output to the DAC. It should then return and repeat the procedure indefinitely (or until a human operator manually resets the system).

The following points should be noted:

(i) The control word required to configure the 8255 is 91H; the base address of the 8255 is 0400H.

(ii) You may assume that overflow will not occur during computational operations.

[20 marks]

Note:

1. The system clock is 20MHz ⇒ 1 clock cycle = 50 ns
2. For example 4(7) represents four clocks for an 8-bit operation and seven clocks for a 16-bit operation.
3. For example 8/4 represents eight clock cycles if jump is taken and four clock cycles is jump is not taken.
4. You may assume an execution time of 10 clock cycles for any memory reference instruction. i.e. reg. to mem. or mem. to reg. (direct, indexed, etc.)

Syntax Abbreviations

Abbreviation	Meaning
accum	One of the accumulators: AX or AL
reg	One of the byte or word registers Byte: AL, AH, BL, BH, CL, CH, DL, DH Word: AX, BX, CX, DX, SI, DI, BP, SP
segreg	One of the segment registers: CS, DS, SS, ES
r/m	One of the general operands: register, memory address, indexed operand, based operand, based-indexed operand
immed	8- or 16-bit immediate value: constant or symbol
mem	One of the memory operands: label, variable, symbol
label	Instruction label
src	Source in string operations
dest	Destination in string operations

Syntax	Action	Clock Cycles
AAA	ASCII adjust for addition	4
AAD	ASCII adjust for division	60
AAM	ASCII adjust for multiplication	83
AAS	ASCII adjust for subtraction	4
ADC *accum,immed*	Add immediate with carry to accumulator	
ADC *r/m,immed*	Add immediate with carry to operand	
ADC *r/m,reg*	Add register with carry to operand	
ADC *reg,r/m*	Add operand with carry to register	
ADD *accum,immed*	Add immediate to accumulator	immed. to reg. = 4
ADD *r/m,immed*	Add immediate to operand	reg. to reg. = 3
ADD *r/m,reg*	Add register to operand	
ADD *reg,r/m*	Add operand to register	
AND *accum,immed*	Bitwise AND immediate with accumulator	
AND *r/m,immed*	Bitwise AND immediate with operand	
AND *r/m,reg*	Bitwise AND register with operand	
AND *reg,r/m*	Bitwise AND operand with register	
* CALL *label*	Call instruction at label	19(23)
* CALL *r/m*	Call instruction indirect	
CBW	Convert byte to word	
CLC	Clear carry flag	
CLD	Clear direction flag	
CLI	Clear interrupt flag	

* Flags not affected.

Syntax	Action	Clock Cycles
CMC	Complement carry flag	2
CMP *accum,immed*	Compare immediate with accumulator	immed. to reg. = 4
CMP *r/m,immed*	Compare immediate with operand	
CMP *r/m,reg*	Compare register with operand	reg. to reg. = 3
CMP *reg,r/m*	Compare operand with register	
CMPS *src,dest*	Compare strings	
CMPSB	Compare strings byte for byte	
CMPSW	Compare strings word for word	
*CWD	Convert word to doubleword	5
DAA	Decimal adjust for addition	4
DAS	Decimal adjust for subtraction	4
DEC *r/m*	Decrement operand	3
DEC *reg*	Decrement 16-bit register	2
DIV *r/m*	Divide accumulator by operand	
* ESC *immed,r/m*	Escape with 6-bit immediate and operand	
*HLT	Halt	2
IDIV *r/m*	Integer divide accumulator by operand	
IMUL *r/m*	Integer multiply accumulator by operand	98 (128)
*IN *accum,immed*	Input from port (8-bit immediate)	
*IN *accum,DX*	Input from port given by DX	8 (12)
INC *r/m*	Increment operand	3
INC *reg*	Increment 16-bit register	2
INT 3	Software interrupt 3 (encoded as one byte)	52 (72)
INT *immed*	Software interrupts 0-255	
INTO	Interrupt on overflow	
IRET	Return from interrupt	32 (44)
JA *label*	Jump on above	Jump is taken = 8
JAE *label*	Jump on above or equal	Jump is not taken = 4
JB *label*	Jump on below	
JBE *label*	Jump on below or equal	
JC *label*	Jump on carry	
JCXZ *label*	Jump on CX zero	
JE *label*	Jump on equal	
JG *label*	Jump on greater	
JGE *label*	Jump on greater or equal	
JL *label*	Jump on less than	
JLE *label*	Jump on less than or equal	
JMP *label*	Jump to instruction at label	
JMP *r/m*	Jump to instruction indirect	
JNA *label*	Jump on not above	
JNAE *label*	Jump on not above or equal	
JNB *label*	Jump on not below	
JNBE *label*	Jump on not below or equal	
JNC *label*	Jump on no carry	
JNE *label*	Jump on not equal	
JNG *label*	Jump on not greater	
JNGE *label*	Jump on not greater or equal	
JNL *label*	Jump on not less than	
JNLE *label*	Jump on not less than or equal	
JNO *label*	Jump on not overflow	
JNP *label*	Jump on not parity	
JNS *label*	Jump on not sign	
JNZ *label*	Jump on not zero	
JO *label*	Jump on overflow	
JP *label*	Jump on parity	
JPE *label*	Jump on parity even	
JPO *label*	Jump on parity odd	

Flags not affected

*Flags not affected

	Syntax	Action	Clock Cycles
*	JS *label*	Jump on sign	8/4
*	JZ *label*	Jump on zero	8/4
	LAHF	Load AH with flags	4
	LDS *r/m*	Load operand into DS	
	LEA *r/m*	Load effective address of operand	
	LES *r/m*	Load operand into ES	
	LOCK	Lock bus	2
	LODS *src*	Load string	
	LODSB	Load byte from string into AL	
	LODSW	Load word from string into AX	
*	LOOP *label*	Loop	17/5
*	LOOPE *label*	Loop while equal	18/6
*	LOOPNE *label*	Loop while not equal	19/5
*	LOOPNZ *label*	Loop while not zero	19/5
*	LOOPZ *label*	Loop while zero	18/6
*	MOV *accum,mem*	Move memory to accumulator	10 (14)
*	MOV *mem,accum*	Move accumulator to memory	10 (14)
*	MOV *r/m,immed*	Move immediate to operand	reg. to reg. = 2
*	MOV *r/m,reg*	Move register to operand	immed. to reg. = 4
*	MOV *r/m,segreg*	Move segment register to operand	seg. to reg. = 2
*	MOV *reg,immed*	Move immediate to register	reg. to seg. = 2
*	MOV *reg,r/m*	Move operand to register	
*	MOV *segreg,r/m*	Move operand to segment register	
*	MOVS *dest,src*	Move string	
*	MOVSB	Move string byte by byte	
*	MOVSW	Move string word by word	
	MUL *r/m*	Multiply accumulator by operand	
*	NEG *r/m*	Negate operand (2's complement)	reg. = 3
*	NOP	No operation	3
*	NOT *r/m*	Invert operand bits (1's complement)	reg. = 3
	OR *accum,immed*	Bitwise OR immediate with accumulator	reg. to reg. = 3
	OR *r/m,immed*	Bitwise OR immediate with operand	immed. to reg. = 4
	OR *r/m,reg*	Bitwise OR register with operand	
	OR *reg,r/m*	Bitwise OR operand with register	
*	OUT DX,*accum*	Output to port given by DX	8 (12)
*	OUT *immed,accum*	Output to port (8-bit immediate)	10 (14)
*	POP *r/m*	Pop 16-bit operand	
*	POP *reg*	Pop 16-bit register from stack	12
*	POP *segreg*	Pop segment register	12
	POPF	Pop flags	12
*	PUSH *r/m*	Push 16-bit operand	
*	PUSH *reg*	Push 16-bit register onto stack	15
*	PUSH *segreg*	Push segment register	14
*	PUSHF	Push flags	14
	RCL *r/m,1*	Rotate left through carry by 1 bit	2
	RCL *r/m,CL*	Rotate left through carry by CL	8 + 4 per bit
	RCR *r/m,1*	Rotate right through carry by 1 bit	2
	RCR *r/m,CL*	Rotate right through carry by CL	8 + 4 per bit
*	REP	Repeat	2
*	REPE	Repeat if equal	2
*	REPNE	Repeat if not equal	2
*	REPNZ	Repeat if not zero	2
*	REPZ	Repeat if zero	2
*	RET [*immed*]	Return after popping bytes from stack	20
	ROL *r/m,1*	Rotate left by 1 bit	2
	ROL *r/m,CL*	Rotate left by CL	8 + 4 per bit
	ROR *r/m,1*	Rotate right by 1 bit	2
	ROR *r/m,CL*	Rotate right by CL	8 + 4 per bit

* Flags not affected

P.T.

Syntax	Action	Clock Cycles
SAHF	Store AH into flags	4
SAL *r/m,1*	Shift arithmetic left by 1 bit	2
SAL *r/m,CL*	Shift arithmetic left by CL	8 + 4 per bit
SAR *r/m,1*	Shift arithmetic right by 1 bit	2
SAR *r/m,CL*	Shift arithmetic right by CL	8 + 4 per bit
SBB *accum,immed*	Subtract immediate and carry flag	reg. to reg.= 3 immed. to reg. = 4
SBB *r/m,immed*	Subtract immediate and carry flag	
SBB *r/m,reg*	Subtract register and carry flag	
SBB *reg,r/m*	Subtract operand and carry flag	
SCAS *dest*	Scan string	
SCASB	Scan string for byte in AL	
SCASW	Scan string for word in AX	
SHL *r/m,1*	Shift left by 1 bit	2
SHL *r/m,CL*	Shift left by CL	8 + 4 per bit
SHR *r/m,1*	Shift right by 1 bit	2
SHR *r/m,CL*	Shift right by CL	8 + 4 per bit
*STC	Set carry flag	2
STD	Set direction flag	2
STI	Set interrupt flag	2
STOS *dest*	Store string	
STOSB	Store byte in AL at string	
STOSW	Store word in AX at string	
SUB *accum,immed*	Subtract immediate from accumulator	reg. to reg. = 3 immed. to reg. = 4
SUB *r/m,immed*	Subtract immediate from operand	
SUB *r/m,reg*	Subtract register from operand	
SUB *reg,r/m*	Subtract operand from register	
TEST *accum,immed*	Compare immediate bits with accumulator	reg. to reg. = 3 immed. to reg. = 5
TEST *r/m,immed*	Compare immediate bits with operand	
TEST *r/m,reg*	Compare register bits with operand	
TEST *reg,r/m*	Compare operand bits with register	
*WAIT	Wait	
*XCHG *accum,reg*	Exchange accumulator with register	3
*XCHG *r/m,reg*	Exchange operand with register	reg. to reg. = 4
*XCHG *reg,accum*	Exchange register with accumulator	
*XCHG *reg,r/m*	Exchange register with operand	
XLAT *mem*	Translate	11
XOR *accum,immed*	Bitwise XOR immediate with accumulator	reg. to reg. = 3 immed. to reg. = 3
XOR *r/m,immed*	Bitwise XOR immediate with operand	
XOR *r/m,reg*	Bitwise XOR register with operand	
XOR *reg,r/m*	Bitwise XOR operand with register	

* Flags not affected