UNIVERSITY OF DUBLIN

TRINITY COLLEGE

Faculty of Engineering, Mathematics & Science
School of Computer Science & Statistics

B.A.(Mod.) Computer Science **Trinity Term 2009**
Junior Freshman Examination

1BA3 – Introduction to Computing

Tuesday 2nd June 2009 Lower Luce Hall 14:00 - 17:00

Dr Jonathan Dukes

Instructions to Candidates:

- Attempt **FOUR** questions
- All questions carry equal marks

Materials permitted for this examination:

- To be accompanied by a **1BA3 Exam Handout** containing **MC68332 Instruction Set Summary** and **Robot Technical Summary**
- Non-programmable calculators are permitted for this examination
- Please indicate the make and model of your calculator on each answer book used

1. Consider a timestamp scheme that represents a time of day as the number of full seconds that have elapsed since the beginning of the day. Design and write a 68332 assembly language program that will convert a 32-bit timestamp based on this scheme to a null-terminated ASCII string representation of the time, in the form "HH:MM:SS AM/PM". The formatted string should be stored in memory.

"HH", "MM" and "SS" represent the time of day in hours, minutes and seconds respectively and must be padded with a leading zero if necessary. "AM" must be displayed if the timestamp represents a time before midday. Otherwise, "PM" must be displayed.

Examples:

Timestamp	ASCII representation
55628	"03:27:08 PM"
36494	"10:08:14 AM"

Your solution must provide the following:

(i) A pseudo-code or English description of your program

(ii) A commented 68332 assembly language listing

(25 marks)

2. (a) Describe in detail the operation of the `link` and `unlk` 68000 assembly language instructions and explain how they are used to implement stack frames. Use appropriate diagrams to illustrate the effect of the `link` and `unlk` instructions on the system stack.

(8 marks)

(b) The following pseudo-code shows how the greatest common divisor (GCD) of two values, *x* and *y*, may be computed recursively.

```
int gcd(x, y)
{
    if (y == 0)
    {
        result = x;
    }
    else
    {
        result = gcd(y, x % y);

        // Note: x % y denotes the remainder of x / y
    }

    return result;
}
```

Translate the above recursive subroutine into a commented 68332 assembly language subroutine that uses the `link` and `unlk` instructions to implement parameter passing using stack frames.

<u>Note:</u> It is assumed that *x* >= *y*. `x % y` denotes the remainder of x divided by y. Information on the DIVU instruction is provided at the back of this examination. Assume that x and y are both unsigned 16-bit values.

(17 marks)

3. (a) For each of the following addressing modes, calculate the effective memory address that would be accessed. Assume that A5 contains $4200, D3 contains 32 and that all accesses are word-size.

(i) `(A5)`
(ii) `-(A5)`
(iii) `8(A5)`
(iv) `16(A5,D3.w)`

(4 marks)

(b) State what effect executing each of the following instructions would have and briefly explain your answer.

(i) `movea.l $4000,A0`
(ii) `lea $4000,A0`
(iii) `movea.l (A0),A0`
(iv) `lea (A0),A0`

(4 marks)

(c) In statistics, the *mode* of a set of sample values is the most frequently occurring value. For example, the mode of the values [2, 1, 4, 3, 4, 4, 1, 4, 1, 5, 8, 4, 8] is 4, since it is the most frequently occurring value.

Design and write a 68332 assembly language subroutine that will determine the mode of an array of unsigned values in the range 0 to 10. Each value is stored in a single byte-size array element. The address of the start of the array and the number of elements in the array must be passed to your subroutine as parameters. Your subroutine should return the most frequently occurring value to the calling program.

Your subroutine must demonstrate the use of the `link` and `unlk` instructions to implement parameter passing and to allocate memory on the stack to maintain a "running count" of the number of occurrences of each of the values between 0 and 10.

Your solution must provide the following:

(i) A pseudo-code description of your solution

(ii) A commented 68332 assembly language listing

(17 marks)

4. Design and write a 68332 assembly language subroutine that will determine whether one two-dimensional matrix, *A*, is a sub-matrix of a second two-dimensional matrix, *B*. Assume that both matrices contain 16-bit elements and that the row- and column-dimensions of matrix A are less than or equal to the row- and column-dimensions of matrix B respectively.

Your solution must provide the following:

(i) A brief English description of your solution

(ii) A brief description of the parameter passing interface used to pass the start address, row-dimension and column dimension of both matrices, *A* and *B*, to your subroutine

(iii) A pseudo-code description of your solution

(iv) A commented 68332 assembly language listing

The example below illustrates two matrices, *A* and *B*, where *A* is a sub-matrix of *B*.

(25 marks)

A

5	2	6	7
1	2	6	4
1	3	4	2

B

1	2	5	2	5	4	8	3	7	1
6	2	1	7	2	9	3	8	4	9
6	9	8	1	5	3	4	2	8	7
5	2	1	5	2	9	3	1	7	9
4	2	5	2	6	7	1	1	2	7
4	2	1	2	6	4	2	7	6	8
9	4	1	3	4	2	2	3	3	4
2	1	2	2	7	3	4	7	8	8
4	3	4	6	9	7	6	1	1	6
2	1	4	4	7	3	3	2	1	2

5. The record format shown below is used to store the examination results of a single student.

DATA	ADDRESS
Student ID	X
Subject 1 mark	X+4
Subject 2 mark	X+6
Subject 3 mark	X+8
Subject 4 mark	X+10
Subject 5 mark	X+12
Subject 6 mark	X+14
Subject 7 mark	X+16
Subject 8 mark	X+18

The results for a group of students can be stored as an array of such records. Design and write an assembly language subroutine that will compute the following information for the exam results for a group of students:

(i) The average mark obtained in each subject across all students

(ii) The average mark obtained by each student across all the subjects

(iii) The number of students who obtained an average mark less than 40

Your solution must include the following:

(i) A detailed description of how you propose passing parameters and results to and from your subroutine. Your description must include details of any additional data structures you propose using. Explain the reasoning behind your proposed parameter passing methods.

(8 marks)

(ii) A commented assembly language listing for your subroutine.

(17 marks)

6. Design and write a 68332 assembly language program that will read a string of arbitrary length from the console and scroll it across the top line of the "Robot" LCD. The string should scroll in on the right-hand end of the display and out on the left-hand end. After the string has scrolled once across the display, your program should read a subsequent string from the console.

Your program may use any of the trap routines available on the "Robot" computers.

Your solution must provide the following:

(i) An English description of your solution

(ii) A pseudo-code description of your solution

(iii) A commented 68332 assembly language listing

(25 marks)

DIVU DIVUL — Unsigned Divide — DIVU DIVUL

Operation: Destination/Source → Destination

Assembler Syntax:
DIVS.W ⟨ea⟩, Dn32/16 → 16r:16q
DIVS.L ⟨ea⟩, Dq32/32 → 32q
DIVS.L ⟨ea⟩, Dr:Dq64/32 → 32r:32q
DIVSL.L ⟨ea⟩, Dr:Dq32/32 →32r:32q

Attributes: Size = (Word, Long)

Description: Divides the unsigned destination operand by the unsigned source operand and stores the unsigned result in the destination. The instruction uses one of four forms.

The word form of the instruction divides a long word by a word. The result is a quotient in the lower word (least significant 16 bits) and a remainder in the upper word (most significant 16 bits) of the destination.

The first long form divides a long word by a long word. The result is a long quotient; the remainder is discarded.

The second long form divides a quad word (in any two data registers) by a long word. The result is a long word quotient and a long word remainder.

The third long form divides a long word by a long word. The result is a long word quotient and a long word remainder.

Two special conditions may arise during the operation:
1. Division by zero causes a trap.
2. Overflow may be detected before instruction completion. If an overflow is detected, the overflow condition code is set and the operands are unaffected.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X	Not affected.
N	Set if quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero occurs.
Z	Set if quotient is zero. Cleared otherwise. Undefined if overflow or divide by zero occurs.
V	Set if division overflow occurs; undefined if divide by zero occurs. Cleared otherwise.
C	Always cleared.

DIVU DIVUL Unsigned Divide DIVU DIVUL

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										EFFECTIVE ADDRESS					
1	0	0	0	REGISTER			0	1	1	MODE			REGISTER		

Instruction Fields:

Register field — Specifies any of the eight data registers. This field always specifies the destination operand.

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dn	000	Reg. number: Dn		(xxx).W	111	000
An	—	—		(xxx).L	111	001
(An)	010	Reg. number: An		#⟨data⟩	111	100
(An) +	011	Reg. number: An				
– (An)	100	Reg. number: An				
(d_{16}, An)	101	Reg. number: An		(d_{16}, PC)	111	010
(d_8, An, Xn)	110	Reg. number: An		(d_8, PC, Xn)	111	011
(bd, An, Xn)	110	Reg. number: An		(bd, PC, Xn)	111	011

NOTE

Overflow occurs if the quotient is larger than a 16-bit signed integer.

DIVU DIVUL Unsigned Divide DIVU DIVUL

Instruction Format (long form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	REGISTER Dq			1	SIZE	0	0	0	0	0	0	0	REGISTER Dr		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register		Addressing Mode	Mode	Register
Dn	000	Reg. number: Dn		(xxx).W	111	000
An	—	—		(xxx).L	111	001
(An)	010	Reg. number: An		#⟨data⟩	111	100
(An) +	011	Reg. number: An				
– (An)	100	Reg. number: An				
(d_{16}, An)	101	Reg. number: An		(d_{16}, PC)	111	010
(d_8, An, Xn)	110	Reg. number: An		(d_8, PC, Xn)	111	011
(bd, An, Xn)	110	Reg. number: An		(bd, PC, Xn)	111	011

Register Dq field — Specifies a data register for the destination operand. The low-order 32 bits of the dividend come from this register, and the 32-bit quotient is loaded into this register.

Size field — Selects a 32 or 64 bit division operation.
0 — 32-bit dividend is in Register Dq.
1 — 64-bit dividend is in Dr:Dq.

Register Dr field — After the division, this register contains the 32-bit remainder. If Dr and Dq are the same register, only the quotient is returned. If Size is 1, this field also specifies the data register that contains the high-order 32 bits of the dividend.

NOTE

Overflow occurs if the quotient is larger than a 32-bit signed integer.