SECTION 1:
The Program Life Cycle
(60 Marks Total)

The questions in Section 1 are based on the following listing file for an MC68000 program. (You do *not* need to understand the program.)

```
 1 0000                          xref    stop
 2 0000   43ED      foo:         lea     array,a1
          0008
 3 0004   222D                   move.l  indx,d1
          0004
 4 0008   242D                   move.l  offset,d2
          0000
 5 000C   0481                   subi.l  #1,d1
          0000
          0001
 6 0012   2619      loop:        move.l  (a1)+,d3
 7 0014   9791                   sub.l   d3,(a1)
 8 0016   2371                   move.l  -4(a1,d2.l),-4(a1)
          20FC
          FFFC
 9 001C   51C9                   dbra    d1,loop
          FFF4
10 0020   4EBA                   jsr     stop
          FFDE
11 0000                         data
12 0000   0000     offset:      dc.l    44
          002C
13 0004   0000     indx:        dc.l    10
          000A
14 0008            array:       ds.l    20
15 0058                         end

                SYMBOL TABLE
                ************

                    CODE
foo    00000000     loop   00000012     stop   00000000

                    DATA
array  00000008     indx   00000004     offset 00000000

                CROSS REFERENCE TABLE
                *********************

array  14#  2
foo    2#
indx   13#  3
loop   6#   9
offset 12#  4
stop   ****  10
```

Question 1. [15 marks]

[5] (i) Briefly, what is the function of the symbol table? You may make references to the listing file above (on page 2) for clarification or examples.

[5] (ii) Why do many assemblers make two passes through a program during the assembly process?

[5] (iii) Briefly, what services must the linker/loader supply for the program shown in the listing file above (on page 2)?

Question 2. [15 marks]

[12] (i) Suppose in the instruction MOVE.L -4(A1,D2.L),-4(A1) in the program of page 2, the source operand is changed to #17.
Give the machine code translation of the new instruction, including all extension words.
[The template for the MOVE instruction is included in Appendix 1 for this exam]

[3] (ii) What is the new program location counter value for the instruction immediately following this MOVE.L instruction?

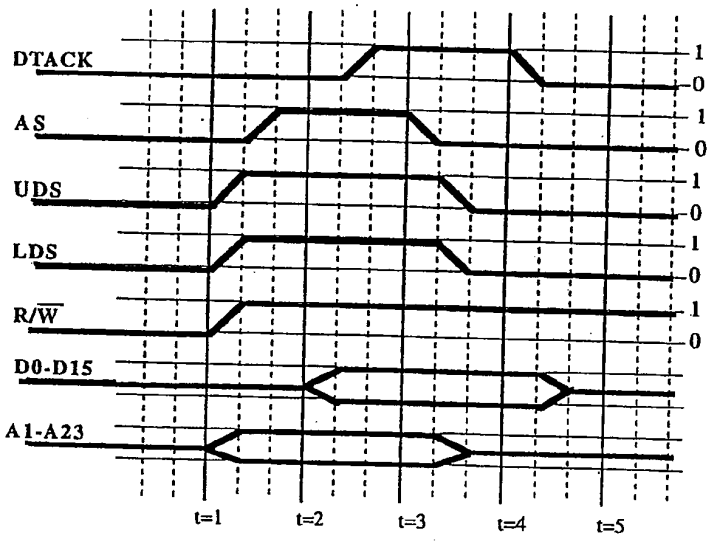
Question 3. [30 marks]

[20] (i) Describe the sequence of events which must take place to fetch, decode, and execute the instruction, given that the program counter contains value $00302014 and the load displacement is $00302000. Clearly specify the order of events, and show the values of the data bus, address bus, program counter, and instruction register.

You may assume register A1 contains $00204024, D3 contains $0000 0001, and memory location $00204024 contains $00000002.

[10] (ii) Relate the sequence of events involved in fetching the opcode word of part (i) to the timing diagram shown below, clearly describing the steps of the transfer protocol.

Timing Diagram



13/8

SECTION 2:
Programming and Stack Handling
(60 marks total)

In Section 2 you will implement a function call using MC68000 assembly language routines, and with Pascal style conventions regarding parameter passing and stack usage/cleaning protocols.

The routine, checklist, takes as input parameters the address of an array, the number of elements in the array, and a (long) integer value. It then checks the elements of the array, indexed from 0 to size-1, looking for the first occurance of the key. The index location of the first occurance of the key is returned, and if no element of the array matches the key then the value "size" is returned. (Pseudo-code is provided in questions 4, 5, and 6.)

Question 4 [12 marks].

Given that the pseudo-code for the function call is

```
index = checklist(array,size,key);
```

and the actual assembly language declarations for the variables (in the calling routine) are:

```
array:  ds.l    20
size:   ds.l    1
key:    ds.l    1
index:  ds.l    1
```

[8] (i) Give the code necessary for the calling routine to place the parameters on the stack and make the subroutine call.

[4] (ii) Give the code necessary for the calling routine to retrieve the return parameter, store the result in the index variable, and perform stack cleaning (if any).

Question 5. [24 marks]
Suppose the skeleton of the checklist routine has the following pseudo-code:

```
long checklist(a, s, k)
long a[], s, k;
{  long i, done;
   done = 0;
   i = s;

   /* body of checklist routine, unknown  for now */

   return(i);
}
```

[6] (i) Give the assembler code necessary to:

1) create the stack frame using the link instruction, including space for local variables i, done.
2) save registers D1-D7, A1-A4 on the stack
3) copy the parameters to local registers/variables, and initialize i, done.

[12] (ii) Draw a picture of what the stack should look like immediately after execution of the link instruction (including everything put on the stack as part of question 4). Show the stack pointer and the frame pointer.

[6] (iii) Give the assembler code necessary at the end of the routine to:
1) return the value in i to the calling program
2) restore the saved registers
3) perform any stack cleaning and unlinking that may be required by the protocol
4) return from the subroutine

Question 6. [24 marks]
The balance of the pseudo-code for the checklist routine is as follows:

```
if (s > 0) {
    i = 0;
    while (done == 0) {
      if (a[i] == k) done = 1;
      else i++;
    }
}
```

[18] (i) Give assembler code for the balance of the checklist routine.

[6] (ii) If the link instruction was not explicitly available, how could you simulate it using 3 standard MC68000 instructions?

SECTION 3:
Architecture
(60 marks total)

Question 7. [12 marks]
[3] (i) Describe the distinguishing features of a RISC system (as opposed to CISC).

[6] (ii) Describe the parameter passing system, discussed in class, which uses a set of 27 registers and a 9 register "window".

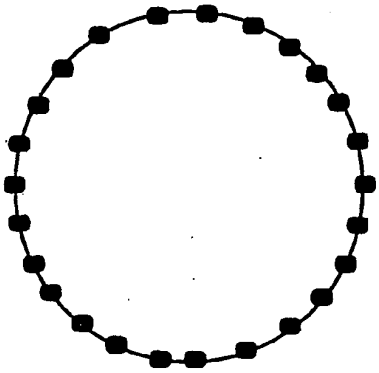
[3] (iii) What are the limitations of such a parameter passing system?

15/00

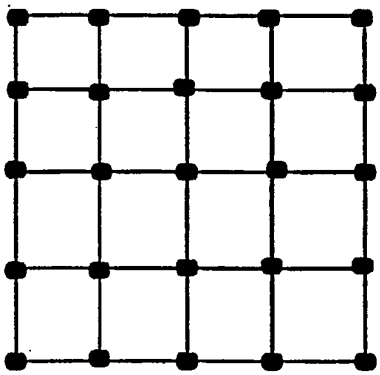
Question 8. [12 marks]

Here are two different interconnection schemes for parallel processors:

Ring network



square mesh network



Suppose each system has N processors, and each processor can send one message to each of its neighbours every time step. (All links are bidirectional.)

[6] (i) For each scheme, how many total interconnections are required?

[6] (ii) For each scheme, what is the longest time it can take for a message to get from processor A to processor B if it goes by the optimal route between A and B?
Assume intermediate processors on the route always pass the message along on the first time cycle after they receive it.
(i.e. what is the farthest apart any two processors can be in the network)

11

Question 9. [12 marks]

Suppose we have a system with 1024 pages of byte-addressable memory, and each page consists of 1024 bytes.

[3] (i) How large an address space is required for the cpu to be able to access any location in memory?

[9] (ii) Suppose we only have a 16-bit address bus. Show how a memory mapping table can be used to give access to 64 pages of memory at any given time.

Question 10. [24 marks]
This question deals with interrupt processing in an MC68000 system.

[2] (i) Name two possible causes of an internal exception, and two possible causes of an external interrupt.

[2] (ii) Describe how an external device communicates an interrupt to the cpu.

[3] (iii) Describe the decision-making process for determining if an external interrupt should be processed.

[5] (iv) Describe the interrupt acknowledgement process.

[8] (v) Suppose acknowledgement comes in the form of a signal on the DTACK pin. What form of interrupt is this? How is an exception vector obtained in this case, and how is it used to gain access to an exception handling routine?

[4] (vi) How can the handling of exception vectors be used to introduce your own exception handling routines?

-- THE END --

Appendix 1

MOVE — Move Data from Source to Destination

Operation : Source → Destination

Assembler Syntax : **MOVE <ea>, <ea>**

Attributes : Size = (Byte, Word, Long)

Description : Moves the content of the source to the destination location. The data is examined as it is moved, and the condition codes set accordingly. The size of the operation may be specified as byte, word, or long.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N Set if the result is negative. Cleared otherwise.
Z Set if result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format :

15	14	13 12	11 10 9	8 7 6	5 4 3	2 1 0
			Destination		Source	
0	0	Size	Register	Mode	Mode	Register

Instruction Fields :

Size field - Specifies the size of the operand to be moved:

01 - byte operation
11 - word operation
10 - long operation

Destination Effective Address field - Specifies the destination location. Only data alterable addressing modes are allowed as shown :

Addr. Mode	Mode	Register
Dn	000	reg. number:An
An	-	-
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	-	-
(d_{16},PC)	-	-
(d_8,PC,Xn)	-	-

Source Effective Address field - Specifies the source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d16,PC)	111	010
(d8,PC,Xn)	111	011

* For byte size operation, address register direct is not allowed.

Notes : 1. MOVEA is used when the destination is an address register. Most assemblers automatically make this distinction.
2. MOVEQ can also be used for certain operations on data registers.

Appendix 2

COPY OF PROGRAM LISTING FROM PAGE 2, SECTION 1

```
 1 0000                         xref      stop
 2 0000   43ED  foo:            lea       array,a1
          0008
 3 0004   222D                  move.l    indx,d1
          0004
 4 0008   242D                  move.l    offset,d2
          0000
 5 000C   0481                  subi.l    #1,d1
          0000
          0001
 6 0012   2619  loop:          move.l    (a1)+,d3
 7 0014   9791                  sub.l     d3,(a1)
 8 0016   2371                  move.l    -4(a1,d2.l),-4(a1)
          20FC
          FFFC
 9 001C   51C9                  dbra      d1,loop
          FFF4
10 0020   4EBA                  jsr       stop
          FFDE
11 0000                        data
12 0000   0000  offset:        dc.l      44
          002C
13 0004   0000  indx:          dc.l      10
          000A
14 0008         array:         ds.l      20
15 0058                        end
```

SYMBOL TABLE
****

CODE

foo	00000000	loop	00000012	stop	00000000

DATA

array	00000008	indx	00000004	offset	00000000

CROSS REFERENCE TABLE
****

array	14#	2
foo	2#	
indx	13#	3
loop	6#	9
offset	12#	4
stop	****	10