Full Name:______________________________________

Email:______________________________________

CS 15-740, Fall 1995

Final Exam

Dec. 11, 1995

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and email address *on every indicated sheet of the exam.*
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 120(!) points.
- If you feel certain details are missing from the problem statement, you may make reasonable assumption, but state them clearly.
- This exam is OPEN BOOK. You may use any books or notes you like. You'll even be allowed to leave the room and do the exam elsewhere, although any form of collaboration is prohibited.

Do not write below this line

P6 & ILP(20)	System Bus(20)	Memory Hier.(20)	Vector Machine(20)	Cilk & NESL(20)	MPI & PVM(20)
1. 8 :_____	4. 5 :__ __	8. 5 :_____	12. 20 :_____	13. 10 :_____	15. 20 :_____
2. 4 :_____	5. 5 :_____	9. 5 :_____		14. 10 :_____	
3. 8 :_____	6. 5 :_____	10. 5 :_____			
	7. 5 :_____	11. 5 :_____			
Total: 100 : _____					

Full Name:_______________________________________

Email:_______________________________________

P6 and Instruction Level Parallelism

Instruction execution on the P6 can be divided into three parts: instruction fetch, out-of-order execution, and instruction retirement. The instruction fetch unit makes sure that instructions are fetched in program order and are logged in the Reorder Buffer (ROB). The out-of-order execution unit allows several instructions to be executed out of order. Instructions are placed in reservation stations, where they wait until both their operands and the appropriate execution unit are available. The results coming out from the arithmetic execution units are written into the ROB, and the results coming out from the address calculation units are written to the memory reorder buffer (MOB).

The ROB can retire up to 3 instructions/cycle. No instruction can be retired until all previous instructions have be retired.

The MOB handles load and store instructions (to the L1 data cache, which is divided into 4 interleaved banks). The load and store instructions will be put into the MOB when their addresses are calculated and the data (for stores) is ready. The MOB will issue a write request to the L1 cache when the corresponding store instruction is retired. After a read request from the MOB, the load data is passed from the cache directly to the ROB.

The data cache can process one load and one store per cycle as long as they access different banks. The cache has a latency of 3 cycles, but it is fully pipelined and outputs one result per cycle. If an access misses the cache, the cache can continue to serve requests while waiting for the miss data to be returned. This type of cache is called a nonblocking cache.

It may be useful to take a look at Figure 1 of the paper "P6 Uses Decoupled Superscalar Design", which was handed out in class.

Problem 1. (8 points):

First, we suppose that we redesign the ROB circuit (call the new ROB "Design 1") so that it can retire an instruction as soon as its result has been computed. It seems to us that this design can't work properly. List the situations in which the new design works incorrectly.

Full Name:____
Email:____

Problem 2. (4 points):

Second, suppose that we redesign the ROB again ("Design 2") so that it can retire instructions whose results are ready and whose input and output operands are not referenced by any previous unretired instructions. Will Design 2 work properly? If not, why not?

Problem 3. (8 points):

Suppose that a cache miss occurs on a read request from the MOB. Can the MOB then continue to issue read and write requests to the cache for subsequent loads and stores?

Suppose that a store instruction is waiting in the MOB to be retired. Can the MOB issue a memory read request to the cache for a load (to a different memory address) that follows the store?

What about a following store?

System Bus Traffic

This problem concerns a proposed design for a personal computer system suitable for high performance, multimedia applications. Your job is to do a general feasibility study of the design, and where appropriate, suggest alternatives. You will need to use your general knowledge of system design and the applications to answer the questions. Don't ask for more details from the instructors—part of your job is to fill in these details.

The system hardware consists of the following:

- A 32-bit RISC CPU, running at 100MHz and 100MIPS with instruction and data caches.
- 64MB of primary memory in the form of 50ns, noninterleaved DRAM.
- A 4GB disk, having a bandwidth of 2MB/s.
- A CDROM drive, having a bandwidth of 300KB/s.
- A graphics display system, displaying images at a resolution of 1024×1024, with either 8-bit or 24-bit color, and a refresh rate of 60 frames per second.
- A "frame grabber" capable of converting NTSC video into bit mapped images with a resolution of 512×512 with 16-bit color and 60 frames per second.
- A system bus connecting the CPU, the memory, the display's frame buffer, the video frame grabber's frame buffer, and the SCSI controller (providing the connection to the hard disk and CDROM drive). The bus can sustain 40MB/s.

Assume for encoding video data, there are three standards:

Grabber The frame grabber output described above.

Quicktime with a resolution of 300×300, 8 bits of color, and a rate of 16 frames per second.

Compressed Quicktime using MPEG compression on the above format to achieve a reduction of 20:1.

For encoding still images, there are two standards:

Slides High resolution (4096×4096) images with 24-bit color.

Compressed Slides Using JPEG compression on the above format to achieve a reduction of 5:1.

When displaying a slide show, assume that a new image is required every 0.5 seconds.

Full Name:_______________________________________

Email:_______________________________________

Problem 4. (5 points):

For each of the 5 data formats described above, calculate the (average) number of bytes to store a single frame, and the bandwidth required to display them properly.

Problem 5. (5 points):

Suppose I would like to be able to execute a program that takes video images from the frame grabber, converts and compresses them into compressed Quicktime format, and stores them on the hard driver, all in real time. What would be the major limitations in the current design in trying to perform this application? What changes would be required to make this application feasible?

Full Name:____

Email:____

Problem 6. (5 points):

Would it be feasible to read compressed Quicktime videos from the CD ROM and display them in real time, with the CPU performing the decompression? What would be the sources and the amount of bus traffic?

Problem 7. (5 points):

Suppose we wish to display a slide show stored on the CDROM as compressed slides. Would this be feasible with the current configuration? What if we were able to preprocess the images and store them on the disk in some other format? Justify your claims.

Memory Hierarchy

The following series of problems concern a hierarchical memory structure with the following features:

- The memory is byte addressable.
- Virtual addresses are 16 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 128 bytes.
- The TLB is 4-way set associative with 16 total entries.
- The cache is direct mapped, with a 4 byte line size and 16 total lines.

In the following tables, all numbers are given in hexadecimal. The contents of the TLB are as follows. Note: Despite the labels in the figure, each 4-way set occupies a row in the table.

TLB

Index	Set 0			Set 1			Set 2			Set 3		
	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	02	13	1	04	32	1	00	28	1	01	0C	0
1	03	2A	1	01	37	0	02	0A	0	02	0C	1
2	01	38	0	00	18	1	08	01	0	03	11	1
3	05	2B	1	04	1A	1	0D	07	0	01	10	1

The page table for the first 16 pages, and the cache contents are as follows

Page Table (Partial)

VPN	PPN	Valid	VPN	PPN	Valid
000	28	1	008	13	1
001	42	0	009	0C	1
002	18	1	00A	07	1
003	0B	1	00B	24	0
004	0C	0	00C	05	0
005	0B	0	00D	2A	1
006	43	1	00E	11	1
007	10	1	00F	11	0

Cache

Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11
1	15	0	55	59	0B	41
2	13	0	12	E2	89	A4
3	22	1	3A	DD	83	11
4	20	1	45	48	0C	A1
5	22	1	23	99	8E	BB
6	2E	0	29	90	E2	59
7	16	1	62	0C	FF	22
8	22	1	5D	13	E3	49
9	13	1	0D	2B	02	3F
A	1F	1	89	38	F4	CC
B	C4	0	40	A7	05	6F
C	B1	1	38	5F	B1	A4
D	26	1	A7	31	37	F8
E	99	0	0D	0C	52	19
F	51	1	88	96	0B	D8

Full Name:_________________________________

Email:_________________________________

Problem 8. (5 points):

A. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

VPN The virtual page number

VPO The virtual page offset

TLBI The TLB index

TLBT The TLB tag

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

B. The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

PPN The physical page number

PPO The physical page offset

CO The byte offset within the cache line

CI The cache index

CT The cache tag

13	12	11	10	9	8	7	6	5	4	3	2	1	0

Full Name:____

Email:____

For the following problems, you are to show how the memory translates a virtual address into a physical and accesses the cache. For the given virtual address, indicate the TLB entry accessed, the physical address, and the cache byte value returned. Indicate whether the TLB misses, whether a page fault occurs, and whether a cache miss occurs. Carry each translation as far as possible, i.e., until there is insufficient data to complete the translation. Mark '—' through entries that cannot be correctly determined.

Problem 9. (5 points):

Virtual address `04C1`

A. Virtual address format

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

B. Address translation

Parameter	Value
VPN	
TLB Index	
TLB Tag	
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	

C. Physical address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0

D. Physical memory reference

Parameter	Value
Byte offset	
Cache Index	
Cache Tag	
Cache Hit? (Y/N)	
Cache Byte returned	

Full Name:_______________________________________

Email:_______________________________________

Problem 10. (5 points):

Virtual address `02F0`

A. Virtual address format

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

B. Address translation

Parameter	Value
VPN	
TLB Index	
TLB Tag	
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	

C. Physical address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0

D. Physical memory reference

Parameter	Value
Byte offset	
Cache Index	
Cache Tag	
Cache Hit? (Y/N)	
Cache Byte returned	

Full Name:_______________________________________

Email:_______________________________________

Problem 11. (5 points):

Virtual address `0569`

A. Virtual address format

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

B. Address translation

Parameter	Value
VPN	
TLB Index	
TLB Tag	
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	

C. Physical address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0

D. Physical memory reference

Parameter	Value
Byte offset	
Cache Index	
Cache Tag	
Cache Hit? (Y/N)	
Cache Byte returned	

Full Name:______________________________________

Email:______________________________________

Vector Machines

Problem 12. (20 points):

In this problem, you will write a short program to perform a permute-and-transpose operation using the DLXV vector instruction set. The input consists of two 64 by 64 matrices, A and P. The matrix A holds integer data. The matrix P specifies the permutation to be performed. Each row, i, of P holds a permutation π_i of the integers 0 through 63. Initially, matrix A is stored at the address contained in register $R1$, and matrix P is stored at the address contained in register $R2$. Both are stored in row-major order. The output is a 64 by 64 matrix B (to be stored at the address in $R3$), in which the ith column of B is equal the ith row of A permuted according to π_i. To make things simple, you may assume that there is no overlap in memory among the matrices A, P, and B. If it helps, you are permitted to use some scratch space in memory, but do not use more than a few rows worth.

Cilk Problem

The following C function performs a recursive (and exhaustive) search of a game tree to determine which player will win if a board game is played perfectly from a particular board position. (You might recall that one of the most successful applications of Cilk was in the *Socrates chess playing program.) You can think about this problem as if the game was tic-tac-toe. There are two players, X and O, who alternately place pieces on a board. The game ends when one of the players wins (e.g., has three in-a-row), or the game is a draw (e.g., the board is full).

```
#define X 0
#define O 1
#define TIE 2
#define NOT_OVER 3
/* This function assumes that position does not represent a final board
** position.  It returns the winner of a game played perfectly from this
** position on, assuming that it is the turn of player.  The value of
** player is either X or O.  The function returns either X, O, or TIE.*/

int search(struct position, int player)
{ int winner,move;
  int best_outcome = 1-player; /* start out assuming a loss for player */
  /* assume there are NUM_MOVES possible moves in this game */
  /* e.g., in tic-tac-toe, NUM_MOVES is 9                   */

  for (move = 0; move < NUM_MOVES; move++)
  { /* some moves might not be legal, e.g., in tic-tac-toe        */
    /* you can't place an X on a square the already has an X or O */

    if (legal_move(position,player,move))
    { /* first see if the game ends immediately with this move */
      /* immediate winner returns X, O, TIE, or NOT_OVER       */
      winner = immediate_winner(make_move(position,player,move));
      if (winner == player) return player;

      /* if not, play the rest of the game after this move     */
      if (winner == NOT_OVER)
      { winner = search(make_move(position,player,move),1-player);
        /* if player can guarantee a win with this move, we're done */
        if (winner == player) return player;
      }
      /* if we get to here, a TIE would be the best so far */
      if (winner == TIE) best_outcome = TIE;
    }
  }
  return best_outcome;
}
```

The following thread is a Cilk implementation of the search function.

```
/* result specifies where to stick result
** position is as before
** player is as before
** move indicates which move to try, between 0 and NUM_MOVES
**    (if move == NUM_MOVES, then finish up)
** best_outcome indicates best outcome of moves 0 through move-1.
** last_outcome indicates outcome of move move-1
*/

thread search(cont int result, struct position, int player,
              int move, int best_outcome, int last_outcome);
{ int winner;
  cont int last;

  /* we're done if we've found a winning move, or if out of moves */

  if ((best_outcome == player) || (move == NUM_MOVES))
    send_argument(result, player);

  if (legal_move(position,player,move))
  {
    winner = immediate_winner(make_move(position,player,move));
    if (winner == player)
      send_argument(result,player);
    if (winner == NOT_OVER)
    {
      spawn_next search(result,position,player,move+1,best_outcome,?last);
      spawn search(last,make_move(position,player,move),1-player,
                   0,player,player);
    }
  }
  else spawn_next(result,position,player,move+1,best_outcome,best_outcome);
}
```

The following NESL function implements the same game-tree search program.

```
function search(position, player) =
let
  legal_moves = {move in [0:NUM_MOVES-1] |
            legal_move(make_move(position,player,move)};
  immediate_wins = {move in legal_moves |
            immediate_winner(make_move(position,player,move)) == player};
  immediate_ties = {move in legal_moves |
            immediate_winner(make_move(position,player,move)) == TIE};
  no_immediate_outcome = {move in legal_moves |
            immediate_winner(make_move(position,player,move)) == NOT_OVER};
in
  if (#immediate_wins > 0) then player;
  else
  let
    delayed_wins = {move in no_immediate_outcome |
              search(make_move(position,player,move)) == player};
  in
    if (#delayed_wins > 0) then player;
    else
      let
        delayed_ties = {move in no_immediate_outcome |
                  search(make_move(position,player,move)) == TIE};
      in
        if (#immedate_ties + #delayed_ties > 0) then TIE;
        else 1-player;
```

Full Name:_______________________________________

Email:_______________________________________

Problem 13. (10 points):

Compare the parallelism in the NESL search function to that in the Cilk search thread.

Problem 14. (10 points):

Even ignoring the differences in the compilers for the C, Cilk, and NESL programs, there will be differences in performance when executed on a sequential machine. Which programs will be faster (or slower) and why?

MPI and PVM

Problem 15. (20 pints):

The Message Passing Interface (MPI) provides several modes for sending messages that Parallel Virtual Machine (PVM) does not. In particular, MPI supports synchronous sends, whereas PVM provides only asynchronous sends. A common way to implement MPI is to write it in PVM. In this problem you will show how to implement the MPI blocking synchronous send function in PVM. A blocking send or receive function blocks until the send or receive is complete. A send is synchronous if it does not complete until the corresponding receive has initiated.

To make things simple, we will limit our attention to sending and receiving messages consisting of a single character. Also, to further simplify things, you may assume that there is a one-to-one correspondence between PVM tids and MPI process numbers. Finally, don't worry about any unusual errors occuring, e.g., out of buffer space.

To solve this problem, you should use the following PVM functions. Note that some of the arguments to these functions, as well as to the MPI functions, have been set to constants to simplify things.

```
/* creates a new empty active send buffer         */
/* returns a bufid (id of new buffer)             */
/* constant PvmDataDefault indicates XDR encoding */

int pvm_initsend(PvmDataDefault);

/* packs bytes into active send buffer */

int pvm_pkbyte(char *cp, int nitem, int stride); /* returns error code */

/* blocking send function; sends message in active send buffer */
/* (not necessary to pass bufid of active send buffer)         */
/* returns error code                                          */

int pvm_send(int tid, int msgtag);

/* blocking receive function                            */
/* creates a new active receive buffer, puts msg in it */
/* returns bufid (id of new buffer)                     */

int pvm_recv(int tid, int msgtag);

/* unpacks bytes in active receive buffer          */
/* (not necessary to pass it bufid of active buffer) */
/* returns error code                              */

int pvm_upkbyte(char *cp, int nitem, int stride);
```

Full Name:____
Email:____

You are to show how to implement the following MPI functions.

```
/* MPI standard blocking send                              */
/*    byte is the byte to send                             */
/*    constant 1 is the number of bytes                    */
/*    constant MPI_BYTE is the type                        */
/*    receiver is the process number of the receiver       */
/*    tag is the message tag                               */
/*    constant MPI_COMM_WORLD indicates flat message space */

MPI_Send(char byte,1,MPI_BYTE,int receiver,int tag,MPI_COMM_WORLD);

/* MPI synchronous blocking send */
MPI_SSend(char byte,1,MPI_BYTE,int receiver,int tag,MPI_COMM_WORLD);

/* MPI blocking receive                             */
/*    put the character that is received into buf       */
/*    returns a status structure -- you can ignore that */

MPI_Recv(void *buf,1,MPI_BYTE,int sender,int tag,MPI_COMM_WORLD,
         struct status *);
```

Note: MPI has only one receive function, and it works with all MPI send commands, including standard, synchronous, ready, buffered modes. You don't have to worry about ready or buffered modes.

Since three of the arguments to the MPI send and receive functions are fixed (and we are ignoring the status argument), you may omit them from the functions you implement. Hence, you need to implement:

```
MPI_Send(char byte, int receiver, int tag);
MPI_SSend(char byte, int receiver, int tag);
MPI_Recv(void *buf, int sender,  int tag);
```

Full Name:____

Email:____