Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Department of Computer Science

Computer Architecture and Systems Programming

252-0061-00

Thursday 2nd February 2012, 14:00-17:00

Last Name : ____________________

First Name : ____________________

Leginr. : ____________________

Rules

- You have 180 minutes for the exam.
- Please write your name and Legi-ID number on all sheets of paper.
- Please write your answers on the exam sheet. Please also use the reverse sides of the exam sheets. If you need more paper, please raise your hand so that we can provide you with additional paper. Write your name and Legi-ID number on those extra sheets of paper.
- Write as clearly as possible and cross out everything that you do not consider to be part of your solution. You must give your answers in either English or German.
- The exam consists of 15 questions. The maximum number of points that can be achieved is 180.
- This exam paper consists of 26 pages in addition to this title page. Please read through the exam paper to ensure that you have all the pages, and if not, please raise your hand.
- You are not allowed to use any written aids in this exam, except for a German-English dictionary and the x86 reference sheet that should be on your desk. If the reference sheet is missing, please raise your hand.

Statement

- If you wish us to publish your results (grade) on a web page, then please sign the following statement.

Signature : ____________________

Name: ______________________________ Leginr: ____________________

Question 1 [20 points]

We are running programs on a machine with the following characteristics:

- Values of type `int` are 32 bits. They are represented in two's complement, and they are right shifted arithmetically. Values of type `unsigned` are 32 bits.
- Values of type `float` are represented using the 32-bit IEEE floating point format, while values of type `double` use the 64-bit IEEE floating point format.

We generate arbitrary values `x`, `y`, and `z`, and convert them to other forms as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to other forms */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

For each of the following C expressions, you should indicate whether or not the expression will *always* yield 1. If so, circle "Y". If not, circle "N". You will be graded on each problem as follows:

- If you circle no value, you get 0 points.
- If you circle the right value, you get 2 points.
- If you circle the wrong value, you get -1 points (so don't just guess wildly).

Expression	Always True?
`(x<y) == (-x>-y)`	Y N
`((x+y)<<4) + y-x == 17*y+15*x`	Y N
`~x+~y+1 == ~(x+y)`	Y N
`ux-uy == -(y-x)`	Y N
`(x >= 0) || (x < ux)`	Y N
`((x >> 1) << 1) <= x`	Y N
`(double)(float) x == (double) x`	Y N
`dx + dy == (double) (y+x)`	Y N
`dx + dy + dz == dz + dy + dx`	Y N
`dx * dy * dz == dz * dy * dx`	Y N

Name: ______________________ Leginr: ______________

Question 2 [16 points]

Consider a **5-bit** two's complement representation.

Fill in the empty boxes in the following table.

Addition and subtraction should be performed based on the rules for 5-bit, two's complement arithmetic.

Number	Decimal Representation	Binary Representation
Zero	0	
n/a	-2	
n/a	9	
n/a	-14	
n/a		`0 1100`
n/a		`1 0100`
TMax		
TMin		
TMin+TMin		
TMin+1		
TMax+1		
−TMax		
−TMin		

Name: ________________________ Leginr: ________________

Question 3 [16 points]

Consider the following 8-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next 3 bits are the exponent. The exponent bias is $2^{3-1} - 1 = 3$.
- The last 4 bits are the fraction.
- The representation encodes numbers of the form: $V = (-1)^s \times M \times 2^E$, where M is the significand and E is the biased exponent.

The rules are like those in the IEEE standard(normalized, denormalized, representation of 0, infinity, and NAN). FILL in the table below. Here are the instructions for each field:

- **Binary:** The 8 bit binary representation.
- **M:** The value of the significand. This should be a number of the form x or $\frac{x}{y}$, where x is an integer, and y is an integral power of 2. Examples include 0, $\frac{3}{4}$.
- **E:** The integer value of the exponent.
- **Value:**The numeric value represented.

  Note: you need not fill in entries marked with "—".

Description	Binary	M	E	Value
Minus zero				-0.0
—	0 100 0101			
Smallest denormalized (negative)				
Largest normalized (positive)				
One				1.0
—				5.5
Positive infinity		—	—	$+\infty$

Question 4 [8 points]

Consider the following assembly representation of a function `foo` containing a `for` loop:

```
foo:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
  movl 8(%ebp),%ebx
  leal 2(%ebx),%edx
  xorl %ecx,%ecx
  cmpl %ebx,%ecx
  jge .L4
.L6:
  leal 5(%ecx,%edx),%edx
  leal 3(%ecx),%eax
  imull %eax,%edx
  incl %ecx
  cmpl %ebx,%ecx
  jl .L6
.L4:
  movl %edx,%eax
  popl %ebx
  movl %ebp,%esp
  popl %ebp
  ret
```

Fill in the blanks to provide the functionality of the loop:

```
int foo(int a)
{
    int i;
    int result = ______________________;


    for( ___________ ;  ___________ ; i++ ) {


        ______________________;


        ______________________;
    }
    return result;
}
```

Question 5 [15 points]

This question concerns the following C code. This program reads a string on standard input and prints an integer in hexadecimal format based on the input string it read.

```
#include <stdio.h>

/* Read a string from stdin into buf */
int evil_read_string()
{
    int buf[2];

    scanf("%s",buf);
    return buf[1];
}

int main()
{
    printf("0x%x\n", evil_read_string());
}
```

Here is the corresponding machine code on a 32-bit Linux/x86 machine:

```
08048414 <evil_read_string>:
 8048414:  55                      push   %ebp
 8048415:  89 e5                   mov    %esp,%ebp
 8048417:  83 ec 14                sub    $0x14,%esp
 804841a:  53                      push   %ebx
 804841b:  83 c4 f8                add    $0xfffffff8,%esp
 804841e:  8d 5d f8                lea    0xfffffff8(%ebp),%ebx
 8048421:  53                      push   %ebx                  address arg for scanf
 8048422:  68 b8 84 04 08          push   $0x80484b8            format string for scanf
 8048427:  e8 e0 fe ff ff          call   804830c <_init+0x50>  call scanf
 804842c:  8b 43 04                mov    0x4(%ebx),%eax
 804842f:  8b 5d e8                mov    0xffffffe8(%ebp),%ebx
 8048432:  89 ec                   mov    %ebp,%esp
 8048434:  5d                      pop    %ebp
 8048435:  c3                      ret

08048438 <main>:
 8048438:  55                      push   %ebp
 8048439:  89 e5                   mov    %esp,%ebp
 804843b:  83 ec 08                sub    $0x8,%esp
 804843e:  83 c4 f8                add    $0xfffffff8,%esp
 8048441:  e8 ce ff ff ff          call   8048414 <evil_read_string>
 8048446:  50                      push   %eax                  integer arg for printf
 8048447:  68 bb 84 04 08          push   $0x80484bb            format string for printf
 804844c:  e8 eb fe ff ff          call   804833c <_init+0x80>  call printf
 8048451:  89 ec                   mov    %ebp,%esp
 8048453:  5d                      pop    %ebp
 8048454:  c3                      ret
```

[Question continues on the next page]

[continued]

This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

- `scanf("%s", buf)` reads an input string from the standard input stream (stdin) and stores it at address `buf` (including the terminating '\0' character). It does **not** check the size of the destination buffer.
- `printf("0x%x", i)` prints the integer i in hexadecimal format preceded by "0x".
- Recall that Linux/x86 machines are Little Endian.
- You will need to know the hex values of the following characters:

Character	Hex value	Character	Hex value
'd'	0x64	'v'	0x76
'r'	0x72	'i'	0x69
'.'	0x2e	'l'	0x6c
'e'	0x65	'\0'	0x00
		's'	0x73

Suppose we run this program on a 32-bit Linux/x86 machine, and give it the string "`dr.evil`" as input on stdin.

Here is a template for the stack, showing the locations of `buf[0]` and `buf[1]`. Fill in the value of `buf[1]` (in hexadecimal) and indicate where `ebp` points just **after** `scanf` returns to `evil_read_string`.

(4 points)

```
                |<- buf[0]->|<-buf[1] ->|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

What is the 4-byte integer (in hex) printed by the `printf` inside main? (2 points)

0x ____________________

Suppose now we give it the input "`dr.evil.lives`" (again on a 32-bit Linux/x86 machine).

List the contents of the following memory locations just **after** `scanf` returns to `evil_read_string`. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

(5 points)

`buf[0] = 0x` ____________________

`buf[3] = 0x` ____________________

[Question continues on the next page]

[continued]

Immediately **before** the `ret` instruction at address `0x08048435` executes, what is the value of the frame pointer register `%ebp`? (4 points)

`%ebp = 0x` ______________

You can use the following template of the stack as *scratch space*. *Note:* this does **not** have to be filled out to receive full credit.

```
               <- buf[0] -><- buf[1] ->
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
```

Question 6 [10 points]

Consider the following incomplete definition of a C struct along with the incomplete code for a function `func` given below.

```
typedef struct node {

  ______________  x;

  ______________  y;

  struct node *next;

  struct node *prev;

} node_t;
```

```
node_t n;

void func() {

  node_t *m;

  m = ______________ ;

  m->y /= 16;

  return;
}
```

When this C code was compiled on an IA-32 machine running Linux, the following assembly code was generated for function `func`.

```
func:
  pushl %ebp
  movl n+12,%eax
  movl 16(%eax),%eax
  movl %esp,%ebp
  movl %ebp,%esp
  shrw $4,8(%eax)
  popl %ebp
  ret
```

Given these code fragments, fill in the blanks in the C code given above. Note that there is a unique answer.

The types must be chosen from the following table, assuming the sizes and alignment given.

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
double	8	4

Name: ______________________________ Leginr: ________________

Question 7 [8 points]

Consider the source code below, where `M` and `N` are constants declared with `#define`.

```
int array1[M][N];
int array2[N][M];

void copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

Suppose the above code generates the following assembly code:

```
copy:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
  movl 8(%ebp),%ecx
  movl 12(%ebp),%eax
  leal 0(,%eax,4),%ebx
  leal 0(,%ecx,8),%edx
  subl %ecx,%edx
  addl %ebx,%eax
  sall $2,%eax
  movl array2(%eax,%ecx,4),%eax
  movl %eax,array1(%ebx,%edx,4)
  popl %ebx
  movl %ebp,%esp
  popl %ebp
  ret
```

What are the values of `M` and `N`?

M = ______________

N = ______________

Name: ______________________________ Leginr: __________________

Question 8 [10 points]

Recall that the standard C library defines the following functions:

```
#include <setjmp.h>

int setjmp(jmp_buf env);
void longjmp(jmp_buf env, int val);
```

Explain what each functions does, including the meanings of their arguments and return values.

(4 points)

`setjmp()`:

`longjmp()`:

Under what circumstances is the first argument to `longjmp(env,val)` valid? (2 points)

[Question continues on the next page]

[continued]

What will be the output (if any) of the following program? (4 points)

```
#include <stdio.h>
#include <setjmp.h>

static jmp_buf buf1;
static jmp_buf buf2;

void second(int i) {
  if (i) {
    longjmp(buf1, 0);
  } else {
    longjmp(buf2, 1);
  }
  printf("second %d\n", i);
}

void first(void) {
  if (!setjmp(buf2)) {
    second(1);
    printf("first-first\n");
  } else {
    printf("first-second\n");
    second(0);
  }
}

int main(int argc, char *argv[]) {
  switch( setjmp(buf1) ) {
  case 0:
    printf("case 0\n");
    first();
    break;
  case 1:
    printf("case 1\n");
    break;
  default:
    printf("default\n");
    second(0);
  }
  return 0;
}
```

Answer:

Question 9

[10 points]

Consider the following function for computing the product of an array of n integers. We have unrolled the loop by a factor of 3.

```
int aprod(int a[], int n)
{
    int i, x, y, z;
    int r = 1;
    for (i = 0; i < n-2; i+= 3) {
        x = a[i]; y = a[i+1]; z = a[i+2];
        r = r * x * y * z; // Product computation
    }
    for (; i < n; i++)
        r *= a[i];
    return r;
}
```

For the line labeled `Product computation`, we can use parentheses to create 5 different associations of the computation, as follows:

```
r = ((r * x) * y) * z; // A1
r = (r * (x * y)) * z; // A2
r = r * ((x * y) * z); // A3
r = r * (x * (y * z)); // A4
r = (r * x) * (y * z); // A5
```

This question is about the performance of the function expressed in terms of the number of cycles per element or CPE.

Assume that the run time of the function for an array of length n, measured in clock cycles, is given by the formula:

$$Cn + K$$

– where C is the CPE.

We measured the 5 versions of the function on an Intel Pentium III. The integer multiplication operation on this machine has a latency of 4 cycles and an issue time of 1 cycle.

[Question continues on the next page]

[continued]

The following table shows some values of the CPE, and other values missing. The measured CPE values are those that were actually observed. “Theoretical CPE” means that performance that would be achieved if the only limiting factor were the latency and issue time of the integer multiplier.

Version	Measured CPE	Theoretical CPE
`A1`	4.00	
`A2`	2.67	
`A3`		$4/3 = 1.33$
`A4`	1.67	
`A5`		$8/3 = 2.67$

Fill in the missing entries. For the missing values of the measured CPE, you can use the values from other versions that would have the same computational behavior. For the values of the theoretical CPE, you can determine the number of cycles that would be required for an iteration considering only the latency and issue time of the multiplier, and then divide by 3.

Question 10 [6 points]

The following table gives the parameters for a number of different caches, where m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, and E is the number of lines per set. For each cache, determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b).

Cache	m	C	B	E	S	t	s	b
1.	32	1024	4	4				
2.	32	1024	4	256				
3.	32	1024	8	1				
4.	32	1024	8	128				
5.	32	1024	32	1				
6.	32	1024	32	4				

Name: ______________________________ Leginr: ________________

Question 11 [12 points]

3M (the company that makes Post-It notes) decides to make Post-Its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square.

3M hires you to determine the efficiency of the following algorithms on a machine with a 2048-byte direct-mapped data cache with 32 byte blocks.

You are given the following definitions:

```
struct point_color {
    int c;
    int m;
    int y;
    int k;
};

struct point_color square[16][16];
register int i, j;
```

You should assume:

- `sizeof(int) = 4`
- `square` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `square`. Variables `i` and `j` are stored in registers.

What percentage of the writes in the following code will miss in the cache?

(4 points)

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].y = 1;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: __________ %

[Question continues on the next page]

[continued]

What percentage of the writes in the following code will miss in the cache?

(4 points)

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[j][i].c = 0;
        square[j][i].m = 0;
        square[j][i].y = 1;
        square[j][i].k = 0;
    }
}
```

Miss rate for writes to `square`: __________ %

What percentage of the writes in the following code will miss in the cache?

(4 points)

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].y = 1;
    }
}
for (i=0; i<16; i++) {
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: __________ %

Name: ______________________ Leginr: ______________

Question 12 [11 points]

The following problem concerns the way virtual addresses are translated into physical addresses.

You should assume:

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Virtual addresses are 16 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 1024 bytes.
- The TLB is 4-way set associative with 16 total entries.

In the following tables, **all numbers are given in hexadecimal**.

The contents of the TLB and the page table for the first 32 pages are as follows:

TLB			
Index	Tag	PPN	Valid
0	8	7	1
	F	6	1
	0	3	0
	1	F	1
1	1	E	1
	2	7	0
	7	3	0
	B	1	1
2	0	0	0
	C	1	0
	F	8	1
	7	6	1
3	8	4	0
	3	5	0
	0	D	1
	2	9	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	2	0	10	1	1
01	5	1	11	3	0
02	7	1	12	9	0
03	9	0	13	7	1
04	F	1	14	D	1
05	3	1	15	5	0
06	B	0	16	E	1
07	D	1	17	6	0
08	7	1	18	1	0
09	C	0	19	0	1
0A	3	0	1A	8	1
0B	1	1	1B	C	0
0C	0	1	1C	0	0
0D	D	0	1D	2	1
0E	0	0	1E	7	0
0F	1	0	1F	3	0

[Question continues on the next page]

Name: ______________________________ Leginr: ________________

[continued]

(3 points)

The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

VPO	The virtual page offset
VPN	The virtual page number
TLBI	The TLB index
TLBT	The TLB tag

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

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:

PPO	The physical page offset
PPN	The physical page number

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

[Question continues on the next page]

Name: ______________________________ Leginr: __________________

[continued]

For the following two virtual addresses, indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs.

If there is a page fault, enter "-" for "PPN" and leave the physical address format empty.

Virtual address: `2F09` (4 points)

Virtual address format (one bit per box)

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

Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

Physical address format (one bit per box)

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

Virtual address: `0C53` (4 points)

Virtual address format (one bit per box)

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

Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

Physical address format (one bit per box)

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

Question 13 [10 points]

Consider a dynamic memory allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes.

The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Five helper routines are defined to facilitate the implementation of `free(void *p)`, and they are given below. For each routine, what it does is explained in the comment above the function definition.

For each routine, fill in the body of the helper routines the code section label that implement the corresponding functionality correctly. There are three choices for each routine.

```
/* given a pointer p to an allocated block, i.e., p is a
   pointer returned by some previous malloc()/realloc() call;
   returns the pointer to the header of the block */
void * header(void* p)
{
  void *ptr;

  ________________;
  return ptr;
}
```

Choices:

1. `ptr=p-1`
2. `ptr=(void *)((int *)p-1)`
3. `ptr=(void *)((int *)p-4)`

```
/* given a pointer to a valid block header or footer,
   returns the size of the block */
int size(void *hp)
{
  int result;

  ________________;
  return result;
}
```

Choices:

1. `result=(*hp)&(~7)`
2. `result=((*(char *)hp)&(~5))<<2`
3. `result=(*(int *)hp)&(~7)`

[Question continues on the next page]

[continued]

```
/* given a pointer p to an allocated block, i.e. p is
   a pointer returned by some previous malloc()/realloc() call;
   returns the pointer to the footer of the block */
void * footer(void *p)
{
  void *ptr;

  ________________;
  return ptr;
}
```

Choices:

1. `ptr=p+size(header(p))-8`
2. `ptr=p+size(header(p))-4`
3. `ptr=(int *)p+size(header(p))-2`

```
/* given a pointer to a valid block header or footer,
   returns the usage of the currect block,
   1 for allocated, 0 for free */
int allocated(void *hp)
{
  int result;

  ________________;
  return result;
}
```

Choices:

1. `result=(*(int *)hp)&1`
2. `result=(*(int *hp)&0`
3. `result=(*(int *)hp)|1`

```
/* given a pointer to a valid block header,
   returns the pointer to the header of previous block in memory */
void * prev(void *hp)
{
  void *ptr;

  ________________;
  return ptr;
}
```

Choices:

1. `ptr = hp - size(hp)`
2. `ptr = hp - size(hp-4)`
3. `ptr = hp - size(hp-4) + 4`

Name: ______________________ Leginr: ______________

Question 14 [17 points]

Describe the operation of the Compare-And-Swap instruction found on some CISC processors like the x86 architecture, and also explain why CAS can be difficult to implement efficiently in hardware.

(4 points)

Some uses of CAS suffer from what is known as the "ABA" problem. Explain what this problem is, and give a typical solution to the problem. (4 points)

[Question continues on the next page]

Name: ______________________________ Leginr: ____________________

[continued]

RISC processors like ARM and MIPS machines do not provide CAS, but instead use two instructions which can be used together to do something very similar to CAS. Give names for these two instructions, and describe the operation of each one.

(4 points)

Using these two instructions, give the rough pseudo code for the following function which performs a Compare-And-Swap operation and returns the result on a 32-bit Linux x86 machine:

(3 points)

```
unsigned int CAS(unsigned int cmp, unsigned int *location, unsigned int val)
{






}
```

[Question continues on the next page]

[continued]

Does the RISC approach to synchronization also suffer from the ABA problem? Why?

(2 points)

Name: ____ Leginr: ______________________

Question 15 [11 points]

In the MSI cache coherency protocol, a line in a local cache can be in one three states: **Invalid**, **Modified**, and **Shared**.

For each of the following transitions, what *local operation* (i.e. something that the local core does) will cause the transition to occur?

(3 points)

Invalid $\rightarrow$ Modified ?

Invalid $\rightarrow$ Shared?

Modified $\rightarrow$ Shared?

For each of the following transitions, what *remote operation* (i.e. something that a different core does) will cause the transition to occur locally?

(3 points)

Modified $\rightarrow$ Invalid?

Modified $\rightarrow$ Shared?

Shared $\rightarrow$ Invalid?

[Question continues on the next page]

[continued]

The MSI protocol is very simple, and performance can be dramatically improved by added more states and more signals. The MESI protocol is one such extension.

What specific extra functionality does the MESI protocol provide, and what is its main advantage over MSI?

(5 points)