Full Name:_____

Datorarkitektur, 2006

Test Exam 2

2006-02-28

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name on the front.
- 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 60 points.
- The aproximate limits for grades on this exam are:
  - To pass (G or 3): 30 points.
  - For grade 4: 43 points.
  - For grade VG: 50 points.
  - For grade 5: 55 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. Good luck!

Problem 1. (6 points):

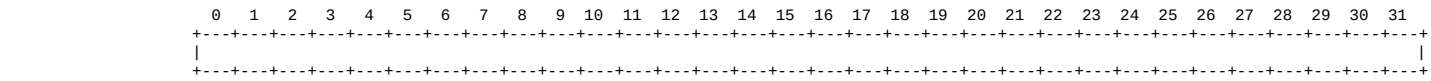
Consider the following datatype definitions on an SPARC machine.

```
typedef struct {
  char  c;
  double *p;
  int i;
  double d;
  short s;
} struct1;
```

```
typedef union {
  char  c;
  double *p;
  int i;
  double d;
  short s;
} union1;
```

A. Using the template below (allowing a maximum of 32 bytes), indicate the allocation of data for a structure of type `struct1`. Mark off and label the areas for each individual element (there are 5 of them). Cross hatch the parts that are allocated, but not used (to satisfy alignment).

Assume the alignment rules discussed in lecture: data types of size x must be aligned on x-byte boundaries. **Clearly indicate the right hand boundary of the data structure with a vertical line**.



B. How many bytes are allocated for an object of type `struct1`?

C. What alignment is required for an object of type `struct1`? (If an object must be aligned on an x-byte boundary, then your answer should be x.)

D. If we define the fields of `struct1` in a different order, we can reduce the number of bytes wasted by each variable of type `struct1`. What is the number of **unused, allocated** bytes in the best case?

E. How many bytes are allocated for an object of type `union1`?

F. What alignment is required for an object of type `union1`? (If an object must be aligned on an x-byte boundary, then your answer should be x.)

Problem 2. (12 points):

In the following questions assume the variables `a` and `b` are signed integers and that the machine uses two's complement representation. Also assume that `MAX_INT` is the maximum integer, `MIN_INT` is the minimum integer, and `W` is one less than the word length (e.g., `W` = 31 for 32-bit integers).

Match each of the descriptions on the left with a line of code on the right (write in the letter). You will be given 2 points for each correct match.

1. One's complement of `a`

_____

2. `a`.

_____

3. `a & b`.

_____

4. `a * 7`.

_____

5. `a / 4` .

_____

6. `(a < 0) ? 1 : -1` .

_____

a. `~(~a | (b ^ (MIN_INT + MAX_INT)))`

b. `((a ^ b) & ~b) | (~(a ^ b) & b)`

c. `1 + (a << 3) + ~a`

d. `(a << 4) + (a << 2) + (a << 1)`

e. `((a < 0) ? (a + 3) : a) >> 2`

f. `a ^ (MIN_INT + MAX_INT)`

g. `~((a | (~a + 1)) >> W) & 1`

h. `~((a >> W) << 1)`

i. `a >> 2`

Problem 3. (12 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		—	—	$+\infty$

Problem 4. (8 points):

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

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

int 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),%ebx
  leal (%ecx,%ecx,8),%edx
  sall $2,%edx
  movl %ebx,%eax
  sall $4,%eax
  subl %ebx,%eax
  sall $2,%eax
  movl array2(%eax,%ecx,4),%eax
  movl %eax,array1(%edx,%ebx,4)
  popl %ebx
  movl %ebp,%esp
  popl %ebp
  ret
```

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

`M` =

`N` =

Problem 5. (3 points):

Consider the following C functions and assembly code:

```
int fun1(int a, int b)
{
    if (a < b)
        return a;
    else
        return b;
}

int fun2(int a, int b)
{
    if (b < a)
        return b;
    else
        return a;
}

int fun3(int a, int b)
{
    unsigned ua = (unsigned) a;
    if (ua < b)
        return b;
    else
        return ua;
}
```

```
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%edx
  movl 12(%ebp),%eax
  cmpl %eax,%edx
  jge .L9
  movl %edx,%eax
.L9:
  movl %ebp,%esp
  popl %ebp
  ret
```

Which of the functions compiled into the assembly code shown?

This next problem will test your understanding of stack frames. It is based on the following recursive C function:

```
int silly(int n, int *p)
{
    int val, val2;

    if (n > 0)
        val2 = silly(n << 1, &val);
    else
        val = val2 = 0;

    *p = val + val2 + n;

    return val + val2;
}
```

This yields the following machine code:

```
silly:
        pushl %ebp
        movl %esp,%ebp
        subl $20,%esp
        pushl %ebx
        movl 8(%ebp),%ebx
        testl %ebx,%ebx
        jle .L3
        addl $-8,%esp
        leal -4(%ebp),%eax
        pushl %eax
        leal (%ebx,%ebx),%eax
        pushl %eax
        call silly
        jmp .L4
        .p2align 4,,7
.L3:
        xorl %eax,%eax
        movl %eax,-4(%ebp)
.L4:
        movl -4(%ebp),%edx
        addl %eax,%edx
        movl 12(%ebp),%eax
        addl %edx,%ebx
        movl %ebx,(%eax)
        movl -24(%ebp),%ebx
        movl %edx,%eax
        movl %ebp,%esp
        popl %ebp
        ret
```

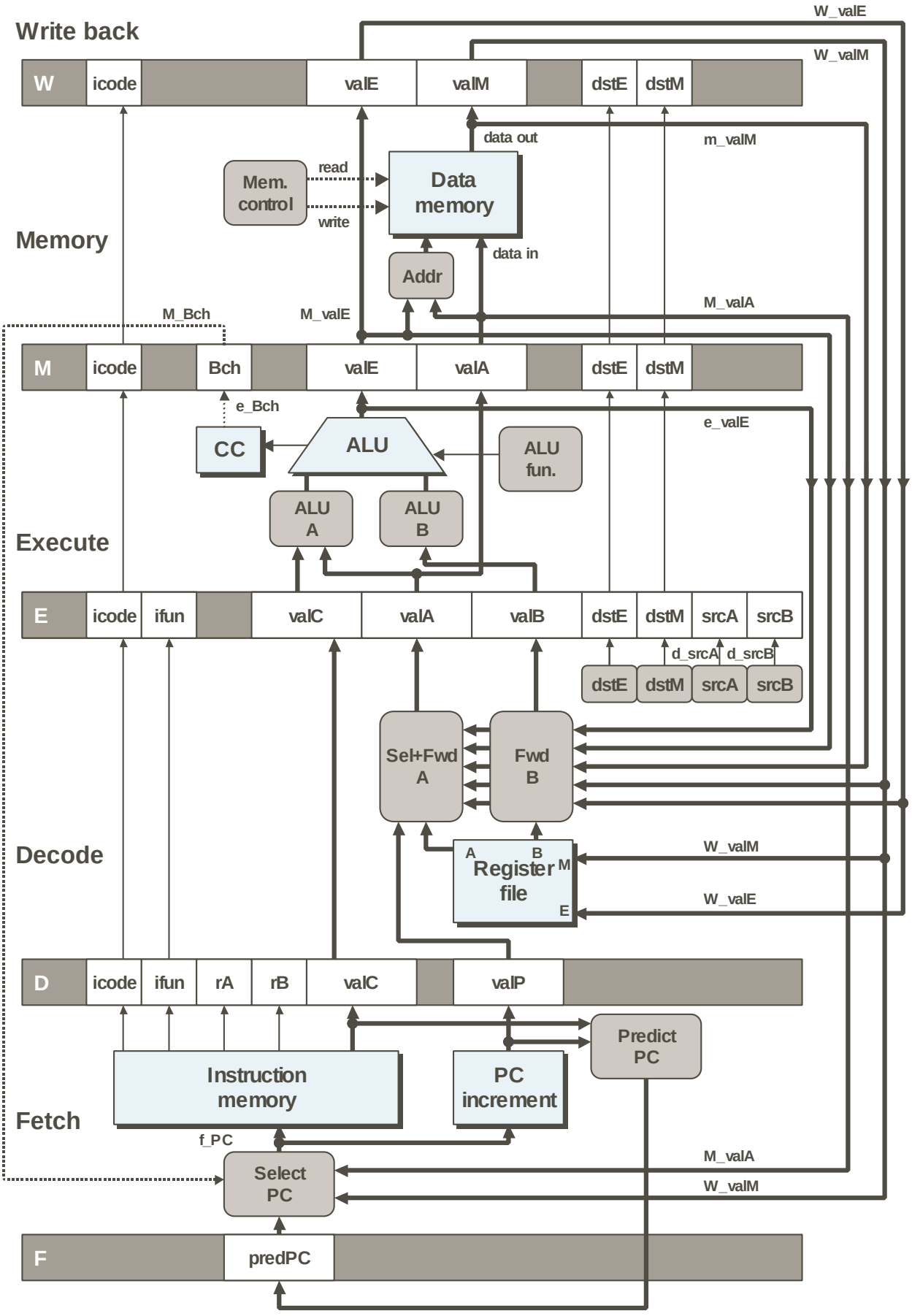
Problem 6. (6 points):

A. Is the variable `val` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored, and why is it necessary to store it on the stack?

B. Is the variable `val2` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored, and why is it necessary to store it on the stack?

C. What (if anything) is stored at `-24(%ebp)`? If something is stored there, why is it necessary to store it?

D. What (if anything) is stored at `-8(%ebp)`? If something is stored there, why is it necessary to store it?


Write back
W_valE
W_valM
W
icode
valE
valM
dstE
dstM
data out
m_valM
Mem. control
read
write
Data memory
Memory
data in
Addr
M_Bch
M_valE
M_valA
M
icode
Bch
valE
valA
dstE
dstM
e_Bch
e_valE
CC
ALU
ALU fun.
ALU A
ALU B
Execute
E
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
d_srcA
d_srcB
dstE
dstM
srcA
srcB
Sel+Fwd A
Fwd B
W_valM
A
B
Register file
M
E
W_valE
Decode
D
icode
ifun
rA
rB
valC
valP
Predict PC
Instruction memory
PC increment
Fetch
f_PC
M_valA
Select PC
W_valM
F
predPC

Problem 7. (13 points):

Modern pipelined machines often have conditionally executed instructions. We want to modify y86 to have the following conditional register to register move instructions: `rrmovl`, `rrmovlle`, `rrmovll`, `rrmovle`, `rrmovlne`, `rrmovlge` and `rrmovlg` that move data from regA to regB on condition respectively: allways (the old `rrmovl`), less or equal, less, equal, not equal, greater or equal and greater.
Example of their use is replacing

```
        andl    %eax, %eax
        jle     LL2
        rrmovl  %ebx, %edx
LL2:
```

with

```
        andl    %eax, %eax
        rrmovlg %ebx, %edx
LL2:
```

A. In the example above, how many cykles would we gain on our SEQ implementation of Y86 if `%eax`≤ 0? If `%eax`> 0? Why?

B. How many cykles would we gain on our PIPE implementation of Y86 if `%eax`≤ 0? If `%eax`> 0? Why?

C. How would you implement these new instructions in PIPE. Draw on the picture of PIPE and describe in words.

D. There may be a data hazard with these new instructions in our PIPE implementation. What is the problem? What can you do about it. Draw on the picture of PIPE and describe in words.