

Here are some sample questions that will be similar to the types of questions that will appear on the final exam. These questions are mainly fill in the blank whereas the final will be multiple choice.

1. Convert the number 57_{10} to the 8-bit binary representation: _____
2. Convert 0x4F2 to the base 10 representation: _____
3. What section of the ELF executable stores the machine code: .text, CODE
4. What interrupt is used to make a system call: int 0x80
5. What is the first instruction executed when entering a function: save caller ebp, pushl ebp
6. How can leave be implemented with mov/pop/push: movl ebp, esp
popl ebp
7. What compiler option is specified to build 32-bit binaries: -m32
8. What instruction move sign extended data from %ax to %ebx: movswl ax,ebx
9. Which of the three cache levels in the processor is the largest: L3
10. What value is returned when fork is called: 0 if child process, else child process id if parent
11. What signal is delivered to your process when you press ctrl-Z: SIGTSTP
12. The runtime complexity for allocation using an implicit free list is: O(N)
13. How byte of storage are required to store the array short s[12]: 24
14. How is the exit value return when a process exits: 24
15. A fully associative cache has how many sets: 1
16. T/F: temporal locality is when proximate data is accessed often: False
17. T/F: A direct mapped cache will have a better hit rate than 4-way Assoc: False
18. What function should be used to install a signal handler: sigaction()
19. What two O.S. calls does the shell use to invoke a program: fork(), execve()

20. Fill in the missing values in the following table.

| Cache | m | C | B | E | S | t | s | b |
|-------|----|-------|----|----|-----|----|----|---|
| 1 | 32 | 65536 | 64 | 16 | | | | |
| 2 | 64 | | | 11 | | | 14 | 6 |
| 3 | 32 | | 64 | 8 | | 17 | | |
| 4 | | 8192 | | | | 18 | 8 | 4 |
| 5 | 32 | | 32 | 1 | 512 | | | |

m = bit in address
 C = cache size (bytes)
 B = block size (bytes)
 E = cache lines per set
 S = number of sets
 t = tag size (bits)
 s = number of bits for set
 b = number of bits for block

For each small program below, show the output of the program. Assume these programs are being compiled and executed on the CSL machines as 32 binaries.

11.

```
#include <stdio.h>

int main(void) {
    int a = 25;
    int b = 19;
    int c = a ^ b;
    printf("%d\n", c);
}
```

Output: _____

12.

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

int main(void) {
    int sum = 0;
    int i = 10;

    while(i) {
        printf("%d\n", i);

        if(i == 2) {
            i++;
            continue;
        }
        if(i == 3) break;
        sum += i;
        i /= 2;
    }
    printf("%d\n", sum);
}
```

Output: _____

13.

```
#include <stdio.h>

int main(void) {
    int value = 0x01020304;
    char *p = (char*) &value;
    for(int i=0; i<4; i++) {
        printf("%02x ", p[i]);
    }
    printf("\n");
}
```

Output: _____

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 00534500: | a7 | d9 | 00 | d8 | 43 | 98 | 8c | b3 | 17 | 00 | 2f | 2d | 58 | 6b | 88 | 99 |
| 00534510: | 48 | 62 | f3 | 23 | 00 | 00 | ed | 0c | 02 | 39 | 00 | 14 | 00 | 5a | 2a | 00 |
| 00534520: | e9 | 00 | c4 | 8a | 9b | 39 | 00 | a1 | d3 | ed | d5 | 00 | 5b | 6c | 51 | 20 |
| 00534530: | 0c | 00 | 6d | 00 | a1 | 19 | 00 | e9 | 81 | c3 | 1e | 00 | 44 | 00 | 00 | c9 |
| 00534540: | de | f6 | 00 | 4f | 00 | 67 | ce | f2 | 00 | 78 | 97 | 34 | 07 | 5e | 00 | 29 |
| 00534550: | 5b | f5 | 46 | 00 | 2e | 2a | 73 | 64 | 06 | ef | 09 | 08 | 4f | f8 | 00 | 00 |
| 00534560: | b7 | 19 | 9a | 00 | cc | 00 | 8c | df | ad | d1 | d1 | be | f0 | e9 | 00 | 00 |
| 00534570: | 2f | f2 | 8c | 00 | 00 | 1b | 4f | 71 | 80 | d7 | d3 | 00 | f1 | 94 | e8 | ab |
| 00534580: | 93 | 73 | 1b | 00 | 39 | 00 | 41 | 35 | 00 | 3c | 00 | ce | 00 | fa | 68 | fc |
| 00534590: | 32 | 01 | 39 | 0f | e5 | 00 | 00 | 36 | 00 | ca | c8 | db | 00 | 1a | 9d | a6 |
| 005345a0: | 43 | dd | e8 | 89 | 00 | 6d | 00 | 00 | 00 | c3 | e8 | c3 | b3 | 60 | 8b | 59 |
| 005345b0: | 00 | 32 | cd | 41 | 73 | f5 | db | a3 | a3 | 79 | c4 | 9a | 5e | 00 | ed | 00 |
| 005345c0: | ee | 20 | 00 | 05 | 00 | 96 | 7c | 08 | 8d | 09 | 66 | 17 | a8 | b2 | cd | 32 |
| 005345d0: | 3e | 00 | 54 | f0 | 00 | 96 | 74 | af | 00 | 82 | 41 | 58 | 1c | 84 | 48 | ec |
| 005345e0: | 39 | 6a | ca | b9 | f0 | 00 | 0b | 00 | 1f | eb | ce | a1 | 74 | 4b | 05 | fc |
| 005345f0: | 77 | d9 | 6d | 00 | 00 | 00 | 00 | 5c | 80 | 45 | 53 | 00 | 50 | ad | 00 | e6 |

Using the map of the contents of memory shown above and assuming that %edx contains 0x00534500 What value will be stored in the %eax register after executing these instructions?

12. movsbl 64(%edx),%eax %eax contains: ff ff ff de

13. movl \$5,%ecx
 movl \$0,%ebx
 movl 0x00534510(%ebx,%ecx,8), %eax %eax contains: 0x00e1c381

14. movl \$16,%ecx
 leal (%edx,%ecx,4), %eax %eax contains:

15. leal 0xf0(%edx),%eax
 movl (%eax), %eax %eax contains:

For each short assembly program, compute the final value stored in the %eax register. The call to printhex32 is a function that will print the value in %eax as a hex value. Essentially this question is asking what is stored in %eax when printhex32 is called.

16. (5pts)

```
main:
    movl $0, %eax
    movl $4, %ebx
loop:
    decl %ebx
    test %ebx,%ebx
    jz done
    leal (%eax,%ebx,4), %eax
    jmp loop
done:
    call printhex32
```

Output: _____

17. (5pts)

```
main:
    movl $3, %eax
    movl $9, %ebx
    movl $16, %ecx
    movl $14, %edx
    xorl %ecx, %edx
    imul %ebx, %eax
    orl %edx, %eax
    andl $31, %eax
    call printhex32
```

Output: _____

18. (5pts)

```
main:
    movl $0, %eax
    movl $8, %ebx
loop:
    decl %ebx
    addl %ebx, %eax
    movl %ebx, %ecx
    andl $5, %ecx
    addl %ecx, %eax
    test %ebx, %ebx
    jne loop
    call printhex32
```

Output: _____

Finish the code below. You need to show what belongs in the 3 location marked HERE.

```
#include <stdio.h>

// the add function takes an array of integers and return the sum of the integers
int add ( /* HERE: you need specify how the function is called */ ) {
    // HERE: fill in the function body.
}

int main() {
    // NOTE: your code should work when a has a different number of elements
    int a[] = {2,3,7,9};

    int sum = add(/* HERE: call your add function*/);

    printf("sum = %d\n", sum);
}
```

It is easiest to rewrite the entire program below (no need to copy the comments)

| N | Power of 2 |
|----|------------|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 |
| 11 | 2048 |
| 12 | 4096 |
| 13 | 8192 |
| 14 | 16384 |
| 15 | 32768 |
| 16 | 65536 |
| 17 | 131072 |
| 18 | 262144 |
| 19 | 524288 |
| 20 | 1048576 |
| 21 | 2097152 |
| 22 | 4194304 |
| 23 | 8388608 |
| 24 | 16777216 |
| 25 | 33554432 |
| 26 | 67108864 |
| 27 | 134217728 |
| 28 | 268435456 |
| 29 | 536870912 |
| 30 | 1073741824 |

x86 cheat sheet

general purpose registers

%eax (%ax,%ah,%al)
%ecx (%cx,%ch,%cl)
%edx (%dx,%dh,%dl)
%ebx (%bx,%bh,%bl)
%esi
%edi
%ebp [base pointer]
%esp [stack pointer]

program counter

%eip
[instruction pointer]

condition codes (CCs)

cf (carry flag)
zf (zero flag)
sf (sign flag)
of (overflowing flag)

data movement

movl src, dst

src or dot can be:

- immediate (e.g., \$0x10 or \$4)
- register (e.g., %eax)
- memory (e.g., an address)

limits:

- dst can never be an immediate
- src or dot (but not both) can be memory

general memory form:

N (register1, register2, C)

which leads to the memory address:

N + register1 + (C * register2)

N can be a large number;

C can be 1, 2, 4, or 8

common shorter forms:

N absolute (reg1=0, reg2=0)
(%eax) register indirect (N=0, reg2=0)
N(%eax) base + displacement (reg2=0)
N(%eax,%ebx) indexed (C=1)

example:

movl 4(%eax), %ebx

takes value inside register %eax, adds 4 to it, and then fetches the contents of memory at that address, putting the result into register %ebx; sometimes called a "load" instruction as it loads data from memory into a register

jump

j dst always jump
je dst jump if equal/zero
jne dst ... not eq/not zero
js dst ... negative
jns dst ... non-negative
jg dst ... greater (signed)
jge dst ... >= (signed)
jl dst ... less (signed)
jle dst ... <= (signed)
ja dst ... above (unsigned)
jb dst ... below (unsigned)

dst is address of code (i.e., jump target)

comparison

cmpl src2, src1
// like computing src1 - src2
cf=1 if carry out from msb
zf=1 if (src1==src2)
sf=1 if (src1-src2 < 0)
of=1 if two's complement under/overflow

testing

testl src2, src1
// like computing src1 & src2
zf set when src1&src2 == 0
sf set when src1&src2 < 0

set

sete dst equal/zero
setne dst not eq/not zero
sets dst negative
setns dst non-negative
setg dst greater (signed)
setge dst >= (signed)
setl dst less (signed)
setle dst <= (signed)
seta dst above (unsigned)
setb dst below (unsigned)

dst must be one of the 8 single-byte reg (e.g., %al)

often paired with movzbl instruction

(which moves 8-byte reg into 32-bit & zeroes out rest)

arithmetic

two operand instructions

addl src,dst dst = dst + src
subl src,dst dst = dst - src
imull src,dst dst = dst * src
sall src,dst dst = dst << src (aka shll)
sarl src,dst dst = dst >> src (arith)
shrl src,dst dst = dst >> src (logical)
xorl src,dst dst = dst ^ src
andl src,dst dst = dst & src
orl src,dst dst = dst | src

one operand instructions

incl dst dst = dst + 1
decl dst dst = dst - 1
negl dst dst = -dst
notl dst dst = ~dst

arithmetic ops set CCs implicitly

cf=1 if carry out from msb
zf=1 if dst==0,
sf=1 if dst < 0 (signed)
of=1 if two's complement (signed) under/overflow