Exam Review

CS351: Systems Programming
Day 15: Oct. 11, 2022

Instructor:
Nik Sultana
Today

- Exam & Grade Structure
- Demo Test review
- Course Review
Exam Structure

- Similar structure & interface to the Demo Test
- Open book/notes/Internet
- Individual exam
- Duration: 45 minutes
  - Exam window opens at 08:30 and closes at 10:00.
  - Don’t spend too long on a question: if stuck, move to the next question and come back to it later.
- 10 questions spanning everything we’ve covered so far.
- Max marks: 120
  i.e., can boost final grade by 5%
- Exam is online but being on campus gives you best chance of getting technical support.
Grade Structure

- Midterm grade != midterm exam
- **Midterm grade** mirrors the final grade structure:
  - 50% labs (i.e., labs 1 and 2 in this case)
  - 50% midterm exam
- On Blackboard you’ll see the Mid-term grade, the mid-term exam marks, and you can already see lab marks.
Today

- Exam & Grade Structure
- Demo Test review
- Course Review
Demo Test: participation

- Establishes significance of analysis on next slides. We’ll analyse how the population performed in each question.
Demo Test: test-level histogram

Most people got >50%!
Demo Test: p/question & p/student results
Demo Test: per-question results

Demo Test questions

#Students

(Sub)Question

Illinois Tech CS351 Fall 2022
Demo Test: Q2

Assume the following values are stored at the indicated memory addresses and registers:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x104</td>
<td>0xAB</td>
</tr>
<tr>
<td>0x108</td>
<td>0x13</td>
</tr>
<tr>
<td>0x10C</td>
<td>0x11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>0x100</td>
</tr>
<tr>
<td>%rcx</td>
<td>0x1</td>
</tr>
<tr>
<td>%rdx</td>
<td>0x3</td>
</tr>
</tbody>
</table>

Provide values for operands indicated in the following table:

<table>
<thead>
<tr>
<th>Operand</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>[1]</td>
</tr>
<tr>
<td>0x104</td>
<td>[2]</td>
</tr>
<tr>
<td>0x108</td>
<td>[3]</td>
</tr>
<tr>
<td>(%rax)</td>
<td>[4]</td>
</tr>
<tr>
<td>4(%rax)</td>
<td>[5]</td>
</tr>
<tr>
<td>9(%rax, %rdx)</td>
<td>[6]</td>
</tr>
<tr>
<td>260(%rcx, %rdx)</td>
<td>[7]</td>
</tr>
<tr>
<td>0xFC(,%rcx, 4)</td>
<td>[8]</td>
</tr>
<tr>
<td>(%rax, %rdx, 4)</td>
<td>[9]</td>
</tr>
</tbody>
</table>
Address Computation Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Address Computation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8 (%rdx)</td>
<td>0xf000 + 0x8</td>
<td>0xf008</td>
</tr>
<tr>
<td>(%rdx, %rcx)</td>
<td>0xf000 + 0x100</td>
<td>0xf100</td>
</tr>
<tr>
<td>(%rdx, %rcx, 4)</td>
<td>0xf000 + 4*0x100</td>
<td>0xf400</td>
</tr>
<tr>
<td>0x80 (%rdx, 2)</td>
<td>2*0xf000 + 0x80</td>
<td>0x1e080</td>
</tr>
</tbody>
</table>
Complete Memory Addressing Modes

- **Most General Form**
  
  $D(Rb,Ri,S) \rightarrow \text{Mem[Reg}[Rb]+S*\text{Reg}[Ri]+D]$

  - $D$: Constant “displacement” 1, 2, or 4 bytes
  - $Rb$: Base register: Any of 16 integer registers
  - $Ri$: Index register: Any, except for $\%\text{rsp}$
  - $S$: Scale: 1, 2, 4, or 8 (*why these numbers?*)

- **Special Cases**

<table>
<thead>
<tr>
<th>(Rb,Ri)</th>
<th>Mem[Reg[Rb]+Reg[Ri]]</th>
</tr>
</thead>
<tbody>
<tr>
<td>D(Rb,Ri)</td>
<td>Mem[Reg[Rb]+Reg[Ri]+D]</td>
</tr>
<tr>
<td>(Rb,Ri,S)</td>
<td>Mem[Reg[Rb]+S*Reg[Ri]]</td>
</tr>
</tbody>
</table>
Demo Test: Q3

Complete this program to make it well-typed:

```java
[A] f(int x, [B] [C]) {
    for (int [D] = 0; i < x; i++) {
        j = (long)(i + x);
    }

    return (char)j;
}
```

For more: Read Chapter 2 of CS:APP3e and practice problems
Demo Test: Q4

Which best describes the type of p, declared below?

```c
char (*p[10])(int *);
```
Points in C

- We encountered pointers several times so far. As with any language: **practice makes perfect!**
- K&R Chapter 5 (can get from library – see announcement on Blackboard and at last lecture).

2. Consider the following C declaration:

```c
int iarr[100];
void *p = iarr;
```

Which of the following expressions is semantically equivalent to “`iarr[50]`”?

- (a) `*(int *)(char *)(p + 50 * sizeof(int))`
- (b) `*(int *)(p + 50 * sizeof(int *))`
- (c) `((int *)(char *)(p + 50))[0]`
- (d) `*(char *)(int *)(p + 50)`

See past exam questions:

Demo Test: Q5

What is wrong with the following structure declaration?

```c
struct foo {
    void *val;
    struct foo *p, *q;
    struct foo x, y;
};
```

For more: Read Chapter 3 of CS:APP3e and practice problems
Structure represented as block of memory
- Big enough to hold all of the fields

Fields ordered according to declaration
- Even if another ordering could yield a more compact representation

Compiler determines overall size + positions of fields
- Machine-level program has no understanding of the structures in the source code

```c
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```
Today

- Exam & Grade Structure
- Demo Test review
- Course review
What did we cover so far?

- Representing data
- Representing programs
- Linking
- Memory
- (+ C and x86_64 toolchains + C review)
Representing data

- **Numeral encoding** (Theory and Practice)
  - Scope: no theorems or proofs since this isn’t a maths course (but helps to understand them)
  - Scope does include two’s complement arithmetic
  - Encoding of integers (signed & unsigned) in C, and max and min values.
  - Conversions/casts between both

- **Encoding other types** (wrt Machine Programming)
  - Arrays, Structs, Unions
  - Alignment
### Two-complement Encoding Example (Cont.)

Given:
- \( x = 15213: 00111011 \ 01101101 \)
- \( y = -15213: 11000100 \ 10010011 \)

<table>
<thead>
<tr>
<th>Weight</th>
<th>15213</th>
<th>-15213</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2048</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8192</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16384</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-32768</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sum</th>
<th>15213</th>
<th>-15213</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sum: 15213 - 15213

From Day 3
Signed vs. Unsigned in C

- **Constants**
  - By default are considered to be signed integers
  - Unsigned if have “U” as suffix
    - `0U`, `4294967259U`

- **Casting**
  - Explicit casting between signed & unsigned same as U2T and T2U
    ```c
    int tx, ty;
    unsigned ux, uy;
    tx = (int) ux;
    uy = (unsigned) ty;
    ```
  - Implicit casting also occurs via assignments and procedure calls
    ```c
    tx = ux;
    uy = ty;
    ```
Array Allocation

Basic Principle

\( T \ A[L] \);

- Array of data type \( T \) and length \( L \)
- Contiguously allocated region of \( L \ast \text{sizeof}(T) \) bytes in memory

\begin{align*}
\text{char} & \text{ string}[12]; \\
\text{int} & \text{ val}[5]; \\
\text{double} & \text{ a}[3]; \\
\text{char} & \ast \text{ p}[3];
\end{align*}
Representing programs

- **Interacting with data**
  (Overlap with previous topic)
  - Arrays, Structs, Unions
  - Alignment

- **Control flow**
  - Branching
  - Procedure calls
  - Loops
Generating Pointer to Structure Member

```c
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};

int *get_ap
(struct rec *r, size_t idx)
{
    return &r->a[idx];
}
```

Generating Pointer to Array Element

- Offset of each structure member determined at compile time
- Compute as `r + 4*idx`

```
# r in %rdi, idx in %rsi
leaq (%rdi,%rsi,4), %rax
ret
```
Alignment Principles

- **Aligned Data**
  - Primitive data type requires $K$ bytes
  - Address must be multiple of $K$
  - Required on some machines; advised on x86-64

- **Motivation for Aligning Data**
  - Memory accessed by (aligned) chunks of 4 or 8 bytes (system dependent)
    - Inefficient to load or store datum that spans quad word boundaries
    - Virtual memory trickier when datum spans 2 pages

- **Compiler**
  - Inserts gaps in structure to ensure correct alignment of fields
Meeting Overall Alignment Requirement

- For largest alignment requirement $K$
- Overall structure must be multiple of $K$

```c
struct S2 {
    double v;
    int i[2];
    char c;
} *p;
```

The diagram illustrates the alignment of the structure variables within the memory layout. Each variable is aligned to a multiple of $K=8$ bytes, ensuring that the overall structure meets the alignment requirement.
Arrays of Structures

- Overall structure length multiple of K
- Satisfy alignment requirement for every element

```c
struct S2 {
    double v;
    int i[2];
    char c;
} a[10];
```
Conditional Branch Example

Generation

```c
long absdiff
(long x, long y)
{
    long result;
    if (x > y)
        result = x - y;
    else
        result = y - x;
    return result;
}
```

```c
unix> gcc -Og -S -fno-if-conversion control.c
```

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>Argument x</td>
</tr>
<tr>
<td>%rsi</td>
<td>Argument y</td>
</tr>
<tr>
<td>%rax</td>
<td>Return value</td>
</tr>
</tbody>
</table>
Procedure Data Flow

Registers

- First 6 arguments
  - %rdi
  - %rsi
  - %rdx
  - %rcx
  - %r8
  - %r9

- Return value
  - %rax

Stack

- Only allocate stack space when needed
  - Arg 7
  - Arg 8
  - Arg n
  - ...
Linking

- Toolchain flow
- Resolution
  - Symbol not found?
  - >1 symbols found?
- Relocation
- Static and Dynamic
Step 1: Symbol Resolution

```
int sum(int *a, int n);

int array[2] = {1, 2};

int main()
{
    int val = sum(array, 2);
    return val;
}

main.c
```

```
int sum(int *a, int n)
{
    int i, s = 0;
    for (i = 0; i < n; i++) {
        s += a[i];
    }
    return s;

    sum.c
```
Linker Symbols

- **Global symbols**
  - Symbols defined by module $m$ that can be referenced by other modules.
  - E.g.: non-`static` C functions and non-`static` global variables.

- **External symbols**
  - Global symbols that are referenced by module $m$ but defined by some other module.

- **Local symbols**
  - Symbols that are defined and referenced exclusively by module $m$.
  - E.g.: C functions and global variables defined with the `static` attribute.
  - Local linker symbols are *not* local program variables.
Global Variables

- Avoid if you can

- Otherwise
  - Use `static` if you can
  - Initialize if you define a global variable
  - Use `extern` if you reference an external global variable
Memory

- Memory hierarchy
- Memory mountain: throughput vs stride vs size
- Cache structure and look-up
Example Memory Hierarchy

- **L0**: CPU registers
  - Hold words retrieved from the L1 cache.

- **L1**: L1 cache (SRAM)
  - Holds cache lines retrieved from the L2 cache.

- **L2**: L2 cache (SRAM)
  - Holds cache lines retrieved from L3 cache.

- **L3**: L3 cache (SRAM)
  - Holds cache lines retrieved from main memory.

- **L4**: Main memory (DRAM)
  - Holds disk blocks retrieved from local disks.

- **L5**: Local secondary storage (local disks)
  - Hold files retrieved from disks on remote servers.

- **L6**: Remote secondary storage (e.g., Web servers)

Storage devices:
- Smaller, faster, and costlier (per byte)
- Larger, slower, and cheaper (per byte)
General Cache Concepts

Cache

Data is copied in block-sized transfer units

Memory

Smaller, faster, more expensive memory caches a subset of the blocks

Larger, slower, cheaper memory viewed as partitioned into “blocks”
E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set
Assume: cache block size 8 bytes

Address of short int:

\[
t \text{ bits} \quad 0...01 \quad 100
\]

find set
Next week: recorded lectures

- **LEC 16** and **LEC 17** will be pre-recorded and circulated on Blackboard.
  - Do not come to SB104 those days – there will not be an in-person lecture.
  - My away-at-a-conference days are marked on the course calendar.
Questions?
Per-lecture feedback

- Better sooner rather than later!
- I can help with issues sooner.
- There is a per-lecture feedback form.
- **The form is anonymous.**
  (It checks that you’re at Illinois Tech to filter abuse, but I don’t see who submitted any of the forms.)
- [https://forms.gle/qoeEbBuTYXo5FiU1A](https://forms.gle/qoeEbBuTYXo5FiU1A)
- I’ll remind about this at each lecture.