**15-213 Introduction to Computer Systems**

# Exam 2

April 5, 2005

Name: **Model Solution**

Andrew User ID: fp

Recitation Section: ______________________

- This is an open-book exam.
- Notes and calculators are permitted, but not computers.
- Write your answer legibly in the space provided.
- You have 80 minutes for this exam.

| **Problem** | **Max** | **Score** |
|---|---|---|
| 1 | **14** | |
| 2 | **18** | |
| 3 | **12** | |
| 4 | **8** | |
| 5 | **12** | |
| 6 | **11** | |
| **Total** | **75** | |

## 1. Symbols and Linking (14 points)

Consider the following two files, `fib1.c` and `fib2.c`:

```
/* fib1.c */
#define MAXFIB 1024

int table[MAXFIB];
int fib(int n);

int main(int argc, char **argv) {
  int n;
  table[0] = 0;
  table[1] = 1;
  argc--; argv++; /* skip command name */
  while (argc > 0) {
    if (sscanf(*argv, "%d", &n) != 1 || n < 0 || n >= MAXFIB) {
      printf ("Error: %s not an int or out of range\n", *argv);
      exit (0);
    }
    printf("fib(%d) = %d\n", n, fib(n));
    argc--; argv++;
  }
}

/* fib2.c */
int* table;

int fib(int n) {
  static int num = 2;
  if (n >= num) {
    int i = num;
    while (i <= n) {
      table[i] = table[i-1] + table[i-2];
      i++;
    }
    num = i;
  }
  return table[n];
}
```

1. (8 points) Fill in the following tables by stating for each name whether it is local or global, whether it is strong or weak, and the section it occupies in that module (`.text`, `.data`, or `.bss`). Cross out any box in the table that does not apply. For example, cross out the first box in a line of the symbol is not in the symbol table, or cross out the second box in a line if the symbol is not global (and therefore neither weak nor strong).

   `fib1.c`

| | Local or Global? | Strong or Weak? | Which segment? |
|---|---|---|---|
| `table` | | | |
| `fib` | | | |
| `num` | | | |

   `fib2.c`

| | Local or Global? | Strong or Weak? | Which segment? |
|---|---|---|---|
| `table` | | | |
| `fib` | | | |
| `num` | | | |

2. (3 points) When the two files are linked together, symbols will be resolved. For each symbol below, show which module it will be defined in (write `fib1` or `fib2` or `not determined`).

| | Defined in module? |
|---|---|
| `table` | |
| `fib` | |
| `num` | |

3. (3 points) The code which is generated by `gcc -o fib fib1.c fib2.c` may not execute correctly. Explain succinctly why.

## 2. Virtual Address Translation (18 points)

We consider a virtual address system with the following parameters.

- The memory is byte addressable.
- Virtual addresses are 20 bits wide.
- Physical addresses are 16 bits wide.
- The page size is 4096 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 16 virtual pages are as follows. If a VPN is not listed in the page table, assume it generates a page fault.

TLB

| Index | Tag | PPN | Valid |
|---|---|---|---|
| 0 | 03 | B | 1 |
| | 07 | 6 | 0 |
| | 28 | 3 | 1 |
| | 01 | F | 0 |
| 1 | 31 | 0 | 1 |
| | 12 | 3 | 0 |
| | 07 | E | 1 |
| | 0B | 1 | 1 |
| 2 | 2A | A | 0 |
| | 11 | 1 | 0 |
| | 1F | 8 | 1 |
| | 07 | 5 | 1 |
| 3 | 07 | 3 | 1 |
| | 3F | F | 0 |
| | 10 | D | 0 |
| | 32 | 0 | 0 |

Page Table

| VPN | PPN | Valid |
|---|---|---|
| 00 | 7 | 1 |
| 01 | 8 | 1 |
| 02 | 9 | 1 |
| 03 | A | 1 |
| 04 | 6 | 0 |
| 05 | 3 | 0 |
| 06 | 1 | 0 |
| 07 | 8 | 0 |
| 08 | 2 | 0 |
| 09 | 3 | 0 |
| 0A | 1 | 1 |
| 0B | 6 | 1 |
| 0C | C | 1 |
| 0D | D | 0 |
| 0E | E | 0 |
| 0F | D | 1 |

1. (4 points) In the four rows below, mark the bits that constitute the indicated part of the virtual address with an **X**. Leave the remaining bits of each row blank.

Virtual Page Number

| | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VPN | | | | | | | | | | | | | | | | | | | | |

Virtual Page Offset

| | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VPO | | | | | | | | | | | | | | | | | | | | |

TLB Tag

| | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TLBT | | | | | | | | | | | | | | | | | | | | |

TLB Index

| | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TLBI | | | | | | | | | | | | | | | | | | | | |

2. (7 points) For the virtual address `0x7E37C`, indicate the physical address and various results of the translation. If there is a page fault, enter "—" for the PPN and Physical Address. All answers should be given in hexadecimal.

Virtual Address (one bit per box)

| 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | |

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

3. (7 points) For the virtual address `0x16A48`, indicate the physical address and various results of the translation. If there is a page fault, enter "—" for the PPN and Physical Address. All answers should be given in hexadecimal.

Virtual Address (one bit per box)

| 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | |

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

## 3. Process Control (12 points)

Consider the following C program. For space reasons, we do not check return codes, so assume that all functions return normally. Also assume that `printf` is unbuffered.

```
void handler(int sig) {
  printf("H\n");
  exit(0);
}

int main() {
  pid_t pid1, pid2;

  signal(SIGUSR1,handler);

  pid1 = fork();
  if (pid1 == 0) {
    pid2 = fork();
    printf("A\n");
    if (pid2 == 0) {
      printf("B\n");
      exit(0);
    }
    printf("C\n");
    kill(pid2,SIGUSR1);
    exit(0);
  }
  if (waitpid(pid1, NULL, 0) > 0) {
    printf("D\n");
  }
  exit(0);
}
```

Mark each column that represents a valid possible output of this program with 'Yes' and each column which is impossible with 'No'.

| | | | | | |
|---|---|---|---|---|---|
| A | A | A | A | A | A |
| A | C | B | A | A | A |
| B | D | H | B | C | C |
| C | H | C | C | B | H |
| D | | D | H | D | D |
| | | | D | | |

## 4. Exceptional Control Flow (8 points)

The following C program computes an array `v` by a call to an external function `init_vector`, sums up the elements of `v`, and prints the result.

```
#include <setjmp.h>
#define VSIZE 1024
double v[VSIZE];

jmp_buf k;

double sum(int n, double *v) {
  int i;
  double x = 0.0;
  for (i = 0; i < n; i++) {
   /* place additional code here */


    x += v[i];
  }
  return x;
}

int main () {
  init_vector(VSIZE, v);
  printf("%f\n", sum(VSIZE,v));
  exit(0);
}
/* put new version of main below */
```

We want to change the `sum` function to indicate an error if any of the elements of the input vector is negative by using a long jump to `k`.

Add a line of code to `sum` in the indicated place and write a new version of `main` that prints the same output if there is no error condition and prints `Illegal vector` if any of the vector elements is negative. Your `main` function must still call `sum`.

## 5. Garbage Collection (12 points)

In this problem we consider a tiny list processing machine in which each memory word consists of two bytes: the first byte is a pointer to the tail of the list and the second byte is a data element. The end of a list is marked by a pointer of `0x00`. We assume that the data element is never a pointer.

1. (6 points) In the first question we consider a copying collector.

   We start with the memory state on the left, where the range `0x10`–`0x1F` is the from-space and the range `0x20`–`0x2F` is the to-space. All addresses and values in the diagram are in hexadecimal.

   Write in the state of memory after a copying collector is called with root pointers `0x12` and `0x14`. You may leave cells that remain unchanged blank.

Before GC

| Addr | Ptr | Data |
|---|---|---|
| 10 | 00 | 00 |
| 12 | 1C | 3F |
| 14 | 1E | 0E |
| 16 | 04 | 44 |
| 18 | 1C | 01 |
| 1A | 14 | 20 |
| 1C | 18 | 02 |
| 1E | 00 | 00 |

After GC

| Addr | Ptr | Data | Addr | Ptr | Data |
|---|---|---|---|---|---|
| 10 | | | 20 | | |
| 12 | | | 22 | | |
| 14 | | | 24 | | |
| 16 | | | 26 | | |
| 18 | | | 28 | | |
| 1A | | | 2A | | |
| 1C | | | 2C | | |
| 1E | | | 2E | | |

2. (6 points) In the second question we consider a mark and sweep collector.

We use the lowest bit of the pointer as the mark bit, because it is normally always zero since pointers must be word-aligned.

Assume the garbage collector is once again called with root pointers `0x12` and `0x14`. Write in the state of memory after the mark phase, and then again after the sweep phase. You may leave cells that remain unchanged blank. Assume that the free list starts at the lowest unoccupied address and is terminated by a NULL (`0x00`) pointer.

Before GC

| Addr | Ptr | Data |
|---|---|---|
| 10 | 00 | 00 |
| 12 | 1C | 3F |
| 14 | 1E | 0E |
| 16 | 04 | 44 |
| 18 | 1C | 01 |
| 1A | 14 | 20 |
| 1C | 18 | 02 |
| 1E | 00 | 00 |

After Marking Phase

| Addr | Ptr | Data |
|---|---|---|
| 10 | | |
| 12 | | |
| 14 | | |
| 16 | | |
| 18 | | |
| 1A | | |
| 1C | | |
| 1E | | |

After Sweep Phase

| Addr | Ptr | Data |
|---|---|---|
| 10 | | |
| 12 | | |
| 14 | | |
| 16 | | |
| 18 | | |
| 1A | | |
| 1C | | |
| 1E | | |

## 6. Cyclone (11 points)

Consider the following C program which initializes a linked list `p0` with a call to `init_list`, then counts the number of positive members in `p0` and prints the result.

```
typedef struct LIST {
  struct LIST *next;
  int data;
} List;

void count_pos(List* p, int* k) {
  int i = 0;
  while (p) {
    if (p->data > 0)
      i++;
    p = p->next;
  }
  *k = i;
}

int main () {
  int k = 0;
  int *w = &k;
  List* p0 = init_list();
  count_pos(p0, w);
  printf ("%d\n", k);
  return 0;
}
```

1. (6 points) When this program is ported to Cyclone, each pointer variable must be considered to see which attributes it should be assigned. Indicate which attributes apply by writing "yes" or "no" in the appropriate box.

   You may assume that `init_list();` returns a pointer to a (possibly empty) linked list, allocated on the heap. Recall that a thin pointer is simply a bounded pointer with bound 1, written as `@numelts(1)`.

| | `@numelts(1)` | `@notnull` |
|---|---|---|
| `w` | | |
| `p0` | | |
| `p` | | |

2. (5 points) Now consider the function `upto (`$n$`, `$p$`)` which allocates a linked list $0, \ldots, n-1$ followed by the tail $p$ and returns a pointer to it. Therefore, if we call it with `upto (`$n$`, NULL)` it will return a pointer to the list $0, \ldots, n-1$.

```
List* upto (int n, List* p) {
  List* q;
  if (n > 0) {
    q = (List *) malloc(sizeof(List));
    q->data = n-1;
    q->next = p;
    return upto(n-1, q);
  } else {
    return p;
  }
}

List* init_list() {
  List* q0 = upto (10, NULL);
  return q0;
}

int main () {
  int k = 0;
  int *w = &k;
  List* p0 = init_list();
  count_pos(p0, w);
  printf ("%d\n", k);
  return 0;
}
```

For each pointer variable, indicate which region it points to. Recall that `'H` is the notation for the global heap region, and that `'f` is the notation for the stack region of function `f`.

| Variable | Region |
|---|---|
| p | |
| q | |
| q0 | |
| w | |
| p0 | |