| ECE 2035 | Programming HW/SW Systems | Spring 2019 |
|---|---|---|
| 3 problems, 6 pages | Exam Two Solutions | 8 March 2019 |

**Problem 1** (2 parts, 24 points) **Storage Allocation, Strings, and Pointers**

**Part A** ( 16 points) **Assuming a 64-bit system with 64-bit memory interface and 64-bit addresses**, answer the following addressing questions. Assume all alignment restrictions imposed by the hardware are obeyed, and the compiler does not add additional alignment restrictions. Note: `int` and `float` are 4 bytes, and `double` is 8 bytes. For each part below, fill in the value of each expression given that the expression in the comment is true. You may find it helpful to sketch memory allocation including slack for each part. Assume variables are allocated in **global memory** in the order they are declared. **Please only write numbers in each answer box.**

**Part A1** (4 pts).

| | | |
|---|---|---|
| `int i; // &i == 1000`<br>`char c[2];`<br>`double x` | `&c[1]` | **1005** |
| | `&x` | **1008** |

**Part A2** (8 pts).

| | | |
|---|---|---|
| `char *s; // &s == 1000`<br>`char d = 'T';`<br>`struct {`<br>`    char c;`<br>`    double y;`<br>`    int j;`<br>`    float z;`<br>`} thing;` | `&d` | **1008** |
| | `&thing` | **1016** |
| | `&thing.y` | **1024** |
| | `sizeof(thing)` | **24** |

**Part A3** (4 pts).

| | | |
|---|---|---|
| `int m; // &m == 1000`<br>`float *q;`<br>`int *p = &m;` | `&q` | **1008** |
| | `p+1` | **1004** |

**Part B (** 8 points) Fill in what is printed after the following C fragment executes?

```
char  *foo(char *s,char *t){
   char *z = s;
   while (*t){
   if (*t == 't' || *t == 'T')
      t++;
   else
      *s++ = *t++;
   }
   *s = *t;
   return z;
}
char a[25];
char b[] = “This cart tis there!”;
printf(”%s\n”,foo(a,b)); //only write exactly what is printed
```

| | |
|---|---|
| What is printed? | **his car is here!** |

**Problem 2** (4 parts, 40 points) **Accessing Inputs, Locals, Arrays**

**Part A** (16 points) Consider the following C code on a 32-bit machine:

```
typedef unsigned char color;
typedef struct {
      color r;
      color g;
      color b;
} pixel;
pixel frame[256][1024];
int i,j;
color c;
…
c = frame[i][j].g // for some i,j  IMPLEMENT THIS LINE
```

Assuming `i,j,frame,` and `&c` are in `$1, $2, $3,` and `$10` respectively, write MIPS code to implement the indicated line. Do not overwrite any given registers, and use additional registers beginning at `$4.` More line are provided than are necessary.

| Label | Instruction | Comment |
| --- | --- | --- |
| | **sll $4, $1, 10** | **# 1024*i** |
| | **add $4, $4, $2** | **# 1024*i + j** |
| | **addi $5, $0, 3** | **# sizeof(pixel) = 3** |
| | **mult $5, $4** | **# \|pixel\|*(1024*i + j)** |
| | **mflo $4** | |
| | **addi $4, $4, $3** | **# + frame** |
| | **lbu $5, 1($4)** | **# frame[i][j].g** |
| | **sb $5, 0($10)** | **# c = frame[i][i].g** |
| | | |
| | | |
| | | |
| | | |

Assuming a 32-bit system, consider the following C fragment:

```
int Rotate (int Pattern[]) {
   int i = 0;
    . . .
      Pattern[i] = Pattern[i+1];     // Part C
    . . .
   return (i);                       // Part D
}
```

**Part B** (6 points) Write a MIPS code fragment to implement the beginning of Rotate's implementation: set Rotate's frame pointer and allocate its locals.

| Label | Instruction | Comment |
|---|---|---|
| | **add $30, $29, $0** | **# set FP** |
| | **addi $29, $29, -4** | **# make room for local i** |
| | **sw $0, 0($29)** | **# initialize i** |
| | | |

**Part C** (12 points) Suppose the input parameter `Pattern` is stored in `Rotate`'s activation frame just above the return value slot (pointed to by the frame pointer $30). Write a MIPS code fragment to implement the line that has the comment "PartC" in the C code above:

```
Pattern[i] = Pattern[i+1];
```

*Do not assume that the variable i is already in a register (read it from the stack).*

| Label | Instruction | Comment |
|---|---|---|
| | **lw $1, -4($30)** | **# load i** |
| | **lw $2, 4($30)** | **# load Pattern base addr** |
| | **sll $3, $1, 2** | **# scale i by \|int\|** |
| | **add $4, $3, $2** | **# Pattern + i*4** |
| | **lw $5, 4($4)** | **# load Pattern[i+1]** |
| | **sw $5, 0($4)** | **# store in Pattern[i]** |
| | | |
| | | |

**Part D** (6 points) Write a MIPS code fragment to store the return value of Rotate, deallocate its locals and return to its caller. In other words, implement the line that has the comment "Part D" in the C code above: `return(i);`

*Do not assume that the variable i is already in a register (read it from the stack).*

| Label | Instruction | Comment |
|---|---|---|
| | `lw   $1, -4($30)` | `# load i` |
| | `sw   $1, 0($30)` | `# store i at RV slot` |
| | `addi $29, $29, 4` | `# deallocate locals` |
| | `jr   $31` | `# return` |
| | | |
| | | |

**Problem 3** (2 parts, 36 points) **Parameter Passing w/ Activation Frames**

The function `Solver` (below left) calls function `Quad` after completing `code block 1`. Write MIPS code that properly calls `Quad`. Include all instructions between `code block 1` and `code block 2`. Symbolically label all required stack entries at the point just before control is transferred to `Quad` and give their values if they are known (below right).

```
int Quad(int x, int *y, int z[])
{
   . . .
}

int Solver(int a) {
   int b;
   int c[] = {4, 2, -1};

   <code block 1>

   c[2] = Quad(a, &b, c);

   <code block 2>

}
```

| | | |
|---|---|---|
| ... | ... | ... |
| 9604 | a | 6 |
| Solver's FP 9600 | XXX | XXX |
| 9596 | b | |
| 9592 | c[2] | -1 |
| 9588 | c[1] | 2 |
| SP 9584 | c[0] | 4 |
| 9580 | **RA** | **N/A** |
| 9576 | **FP** | **9600** |
| 9572 | **a** | **6** |
| 9568 | **&b** | **9596** |
| 9564 | **c** | **9584** |
| 9560 | **RV** | |
| 9556 | | |
| 9552 | | |

| label | instruction | comment |
|---|---|---|
| | **addi $29, $29, -24** | # Allocate activation frame |
| | **sw $31, 20($29)** | # Preserve bookkeeping info |
| | **sw $30, 16($29)** | |
| | **lw $1, 4($30)** | # Push inputs: **load a** |
| | **sw $1, 12($29)** | # **store a** |
| | **addi $1, $30, -4** | # **compute &b** |
| | **sw $1, 8($29)** | # **store &b** |
| | **addi $1, $30, -16** | # **compute c base address** |
| | **sw $1, 4($29)** | # **store c base address** |
| | jal Quad | # Call Quad |
| | **lw $31, 20($29)** | # Restore bookkeeping info |
| | **lw $30, 16($29)** | |
| | **lw $1, 0($29)** | # Read return value |
| | **sw $1, -8($30)** | # Store return value in c[2] |
| | **addi $29, $29, 24** | # Deallocate activation frame |