# Exam 1 full

Saturday, October 24, 2020 11:31 PM

## Problem 1. (10 points):

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

```
int mat1[M][N];
int mat2[N][M];

int sum_element(int i, int j)
{
  return mat2[i][j] +=  mat1[j][i];
}
```

A. Suppose the above code generates the following assembly code:

```
sum_element:
        movslq  %edi, %rdi           i
        movslq  %esi, %rsi           j
        leaq    0(,%rdi,8), %rax     return = 8i
        subq    %rdi, %rax           return = return - i   (ret = 7i)
        leaq    (%rdi,%rax,4), %rdx  rdx = i + (4·ret)
        leaq    (%rsi,%rsi,8), %rax  ret = j + (8j)
        leaq    (%rsi,%rax,2), %rax  ret = (j + 2ret)
        addq    %rsi, %rdx           rdx = rdx + j
        leaq    (%rax,%rdi), %rdi    i = (ret + i)
        movl    mat2(,%rdx,4), %eax  ret[31:0] = Mat2 + (4i)
        addl    mat1(,%rdi,4), %eax  ret[31:0] = ret[31:0] + Mat1 + 4i
        movl    %eax, mat2(,%rdx,4)
        ret                          Mat2 + 4(Rdx) = Ret[31:0]
```

Rdi = i Rsi = j

| | Rdi | Rsi | Rax | Rdx |
|---|---|---|---|---|
| 1: | i | | | |
| 2: | | j | | |
| 3: | | | 8i | |
| 4: | | | 7i | |
| 5: | | | | i + 4(7i) |
| 6: | | | 9j | |
| 7: | | | 2(9j)+j | |
| 8. | | | | i + 4(7i) + j |
| 9: | 2(9j)+j+i | | | |

19j + i 29i + j

What are the values of M and N?

M = 29

N = 19

mat2[i][j] = 29i + j

Mat1[j][i] = 19j + i

j = 29

i = 19

## Problem 2. (10 points):

Consider the following assembly code for a C `for` loop:

```
decode_me:
   1  cmpl   %esi, %edi
   2  jle    .L5
   3  leal   (%rdi,%rdi), %edx
   4  movl   $1, %eax
   5  subl   %esi, %edx
.L4:
   6  subl   $1, %edi
   7  addl   $4, %esi
   8  addl   %edx, %eax
   9  subl   $6, %edx
  10  cmpl   %esi, %edi
  11  jg     .L4
  12  addl   $46, %eax
  13  ret
.L5:
  14  movl   $47, %eax
  15  ret
```

| | y | x | temp | result |
|---|---|---|---|---|
| 1 | Compare | | | |
| 2 | | | | |
| 3 | | | 2 x | |
| 4 | | | | 1 |
| 5 | | | 2x − y | |
| 6 | | x−− | | |
| 7 | y + 4 | | | |
| 8 | | | | result + temp |
| 9 | | | temp − 6 | |
| 10 | compare | | | |
| 11 | | | | |
| 12 | | | | result +46 |
| 13 | | | | |
| 14 | | | | result = 47 |
| 15 | | | | |

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables x, y, and result in your expressions below — *do not use register names.*)

```
int decode_me(int x, int y)
{
   int result;

   for (result = 1; __y < x__; x--,y = y + 4 ) {
   __result += (2x-y)__;

   }

   return __result +46__;
}
```

a=rdi b=rsi b*=rax temp=rbx

## Problem 3. Stack Discipline (20 points)

Examine the following recursive function:

```
long sunny(long a, long *b) {
  long temp;
  if (a < 1) {
    return *b - 8;
  } else {
    temp = a - 1;
    return temp + sunny(temp - 2, &temp);
  }
}
```

Here is the x86_64 assembly for the same function:

```
0000000000400536 <sunny>:
  400536:        test    %rdi,%rdi
  400539:        jg      400543 <sunny+0xd>
  40053b:    1   mov     (%rsi),%rax
else 40053e: 2   sub     $0x8,%rax
  400542:        retq
  400543:        push    %rbx           save prev rbx
  400544:    3   sub     $0x10,%rsp
  400548:    4   lea     -0x1(%rdi),%rbx
  40054c:    5   mov     %rbx,0x8(%rsp)
  400551:    6   sub     $0x3,%rdi
  400555:    7   lea     0x8(%rsp),%rsi
  40055a:        callq   400536 <sunny>
  40055f:    8   add     %rbx,%rax
  400562:    9   add     $0x10,%rsp
  400566:        pop     %rbx
  400567:        retq
```

Breakpoint

We call **sunny** from **main()**, with registers %**rsi** = **0x7ff**...**ffad8** and %**rdi** = 6. The value stored at address **0x7ff**...**ffad8** is the long value 32 (0x20). We set a breakpoint at "**return *b - 8**" (i.e. we are just about to return from **sunny()** without making another recursive call). We have executed the **sub** instruction at **40053e** but have not yet executed the **retq**.

**Fill in the register values on the next page** and **draw what the stack will look like <u>when the program hits that breakpoint</u>**. Give both a description of the item stored at that location and the value stored at that location. If a location on the stack is not used, write "unused" in the Description for that address and put "-----" for its Value. You may list the Values in hex or decimal. Unless preceded by **0x** we will assume decimal. It is fine to use **f**...**f** for sequences of **f**'s as shown above for %**rsi**. Add more rows to the table as needed. <u>Also, fill in the box on the next page to include the value this call to **sunny** will *finally* return to **main**.</u>

| | a rdi | b rsi | temp rbx | b* rax |
|---|---|---|---|---|
| 1 | 6 | fad8 | 4 | 32 |
| 2 | | | | 24 |
| 3 | | | | |
| 4 | | | 5 | |
| 5 | | | | |
| 6 | 3 | | | |
| 7 | | fac0 | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

2 Sunny(3, fac0)

| | a rdi | b rsi | temp rbx | b* rax |
|---|---|---|---|---|
| 1 | 3 | fac8 | | 5 |
| 2 | | | | -3 |
| 3 | | | | |
| 4 | | | 2 | |
| 5 | | | | |
| 6 | 0 | | | |
| 7 | | faa0 | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

3 Sunny(0, faa0)

| a rdi | b rsi | temp rbx | b* rax |
|---|---|---|---|
| 0 | 9b0 | | 2 |
| | | | -6 |

| Register | Original Value | Value **at Breakpoint** |
|---|---|---|
| **rsp** | **0x7ff...ffad0** | 0x7f...fa90 |
| **rdi** | **6** | 0 |
| **rsi** | **0x7ff...ffad8** | 0x7f... faa0 |
| **rbx** | **4** | 2 |
| **rax** | **5** | -6 |

DON'T FORGET → What value is **finally** returned to **main** by this call? 1

| Memory address on stack | Name/description of item | Value |
|---|---|---|
| 0x7fffffffffffad8 | Local var in **main** | 0x20 |
| 0x7fffffffffffad0 | Return address back to **main** | 0x400827 |
| push 0x7fffffffffffac8 | rbx (saved) | 4 |
| mov 0x7fffffffffffac0 | temp | 5 |
| 0x7fffffffffffab8 | | |
| calq 0x7fffffffffffab0 | return address | 0x400055f |
| push 0x7fffffffffffaa8 | rbx (saved) | 5 |
| mov 0x7fffffffffffaa0 | temp | 2 |
| 0x7fffffffffffa98 | | |
| calq 0x7fffffffffffa90 | return address | 0x400055f |
| 0x7fffffffffffa88 | | |
| 0x7fffffffffffa80 | | |
| 0x7fffffffffffa78 | | |
| 0x7fffffffffffa70 | | |
| 0x7fffffffffffa68 | | |
| 0x7fffffffffffa60 | | |

**Problem 4. (4+6 points):**

A. Consider an implementation of a processor where the combinational circuit latency is $\alpha$ ns (nanosecond). You are going to pipeline this implementation, and in your system the latency of each pipeline register is $\beta$ ns. If you are to divide this entire combinational circuit in $k$ stages, what is the throughput of your pipelined implementation? What about the total latency of a single instruction?

throughput: $\frac{1}{(\beta + \alpha k)}$

latency = $\beta$

K stages

Comb = $\alpha$

Latency: $K\beta + \alpha$

B. Now, assume that the parameters $\alpha$, $\beta$ and $k$ described above have the following relation:

$$\beta = \begin{cases} \alpha/10 & \text{if } k \leq 5, \\ \alpha/10 + k/50 \bullet 9\alpha/10 & \text{if } k > 5. \end{cases} \quad (1)$$

Based on the equation above, clearly present an analysis when would you pipeline this processor. Please consider three different values of $k$ in your analysis: (i) $k$ = 5; (ii) $k$ = 50; and (iii) $k$ = 500.

pipeline when $\beta < \alpha$

K=5: Pipeline

$\beta = \frac{\alpha}{10} = .1\alpha$

K=50: don't Pipeline

$\beta = \frac{\alpha}{10} + \frac{50}{50} \cdot \frac{9\alpha}{10} = \frac{\alpha}{10} + \frac{9\alpha}{10} = \frac{10\alpha}{10} = \alpha$

K = 500: don't Pipeline

$\beta = \frac{\alpha}{10} + \frac{500}{50} \cdot \frac{9\alpha}{10} = \frac{\alpha}{10} + 9\alpha = 9.1\alpha$