CS 356 Processor Organization

Redekopp

Name: _____

Score: _____

1.) Given a loop such as that shown below with the given assembly translation.

```
void f1(int* A, int* B, int N)
for( ; N != 0; A--,B--,N--){
  temp = *A;
  *A = temp + *B + 9;
  *B = temp;
}
```

```
L1:  ld (%rdi),%eax    ; load temp=*A
     ld (%rsi),%ebx    ; load *B
     add %eax,%ebx     ; add temp+*B
     add $9,%ebx       ; add 9
     st  %ebx,(%rdi)   ; store *A
     st  %eax,(%rsi)   ; store *B
     add $-4,%rdi      ; dec. A ptr.
     add $-4,%rsi      ; dec. B ptr.
     add $-1,%rdx
     jne $0,%rdx,L1 ; loop
```

a.) (15 pts.) Schedule the code for our 2-way VLIW (static multiple issue) superscalar discussed in lecture (1 ALU/Branch slot and 1 load/store slot). You may move code up or down and modify it if it helps scheduling. You cannot use register renaming or loop unrolling in this part. Once finished calculate the IPC.

Note: ld's require a 1 cycle stall before a dependent ALU instruction can be issued.

ALU / Branch	Load / Store
`add $-4,%rdi`	`ld (%rdi),%eax`
`add $-4,%rsi`	`ld (%rsi),%ebx`
`add $-1,%rdx`	`st  %eax,4(%rsi)`
`add %eax,%ebx`	
`add $9,%ebx`	
`jne $0,%rdx,L1`	`st  %ebx,4(%rdi)`

IPC = 10 instructions / ____6 _____ clocks = ___1.67___ IPC

b.) (25 pts.) Now unroll the loop for an additional iteration (2 iterations per body) and use register renaming where appropriate. When performing register renaming, use register numbers $15, $16, $17… in that order so that everyone's answer will hopefully be more uniform. Schedule the code for our 2-way VLIW (static multiple issue) superscalar discussed in lecture (1 ALU/Branch slot and 1 load/store slot). You may modify the execution order (up or down) and change lw/sw offsets if it helps scheduling. Assume full forwarding. Once finished calculate the IPC.

```
L1:  ld (%rdi),%eax    ; load temp=*A
     ld (%rsi),%ebx    ; load *B
     add %eax,%ebx     ; add temp+*B
     add $9,%ebx       ; add 9
     st  %ebx,(%rdi)   ; store *A
     st  %eax,(%rsi)   ; store *B
     ld  -4(%rdi),%r8d
     ld  -4(%rsi),%r9d
     add %r8d,%r9d
     add $9,%r9d
     st  %r9d,-4(%rdi)
     st  %r8d,-4(%rsi)
     add $-8,%rdi      ; dec. A ptr.
     add $-8,%rsi      ; dec. B ptr.
     add $-2,%rdx
     jne $0,%rdx,L1 ; loop
```

ALU / Branch	Load / Store
add $-8,%rdi	ld (%rdi),%eax
add $-8,%rsi	ld (%rsi),%ebx
add $-2,%rdx	ld +4(%rdi),%r8d
add %eax,%ebx	ld +4(%rsi),%r9d
add $9,%ebx	st %eax,8(%rsi)
add %r8d,%r9d	st %ebx,8(%rdi)
add $9,%r9d	st %r8d,4(%rsi)
jne $0,%rdx,L1	st %r9d,4(%rdi)

IPC = 16 instructions / ____8_____ clocks = ___2_____ IPC

2.) (12 pts.) Given the code below, perform explicit register renaming to solve all WAW, WAR hazards present in the original code. When performing register renaming, use register numbers %r8, %r9, %r10… in that order so that everyone's answer will hopefully be more uniform.

ld 0(%rdi), %rax
add %rbx,%rax
sub %rdx,%rcx
ld 0(%rsi),%rbx
sub %rbx,%rdx
mov %rsp,%rsi
add %rcx,%rsi
	ld 0(%rdi), %rax
	add %rbx,%rax
	sub %rdx,%rcx
	ld 0(%rsi),%r8
	sub %r8,%rdx
	move %rsp,%r9
	add %rcx,%r9

3.) (5 pts.) Given the code below, assume the first 'ld' instruction stalls due to a cache miss. Assuming an out-of-order, **dynamically** scheduled processor (that performs automatic register renaming), which instructions would be allowed to execute (i.e. are independent) and which instructions would need to stall due to the 'ld'.

Code	Circle the correct answer
ld 0(%rdi), %rax	CACHE MISS
add %rbx,%rax	**Stall** / Execute
sub %rax,%rcx	**Stall** / Execute
ld 0(%rsi),%rbx	Stall / **Execute**
sub %rbx,%rdx	Stall / **Execute**
add %rcx,%rsi	**Stall** / Execute