1 Part A

1.1 sum.ys Iteratively sum linked list elements

Solution The struct ELE used in the sum_list() function shown on listing 1.

```
/* $begin examples */
/* linked list element */
typedef struct ELE {
    long val;
    struct ELE *next;
} *list_ptr;
```

Listing 1: ELE.c

The function sum_list() written is C displayed on listing 2.

```
/* sum_list - Sum the elements of a linked list */
long sum_list(list_ptr ls)
{
    long val = 0;
    while (ls) {
  val += ls->val;
  ls = ls->next;
    }
    return val;
}
```

Listing 2: sum_list.c

Full example of sum.ys with stack initialization, test data, and main function depicted on lising 3 3.

```
# Execution begins at address 0
    .pos 0
    irmovq stack, %rsp                     # Set up stack pointer
    call main                              # Execute main program
    halt                                   # Terminate program

# Sample linked list
    .align 8
ele1:
    .quad 0x00a
    .quad ele2
ele2:
    .quad 0x0b0
    .quad ele3
ele3:
    .quad 0xc00
    .quad 0

main:
    irmovq ele1,%rdi
    call sum_list                           # sum_list(ele1)
    ret

# long sum_list(list *ls)
# ls in %rdi
sum_list:
    irmovq $0,%rax
    andq %rdi,%rdi
    je test
    loop:
    mrmovq (%rdi), %rsi
    addq %rsi,%rax
    mrmovq 8(%rdi),%rdi
    andq %rdi,%rdi
    jne loop
    test: ret

# Stack starts here and grows to lower addresses
    .pos 0x200
stack:
```

Listing 3: sum.ys

1.2 rsum.ys Recursively sum linked list elements

Solution The same ELE struct from listing 1 used int the rsum_list() function. The C version of rsum_list() is on listing 4.

```
/* rsum_list - Recursive version of sum_list */
long rsum_list(list_ptr ls)
{
    if (!ls)
  return 0;
    else {
  long val = ls->val;
  long rest = rsum_list(ls->next);
  return val + rest;
    }
}
```

Listing 4: rsum_list.c

Full version of rsum.ys is on listing 5.

```
# Execution begins at address 0
    .pos 0
    irmovq stack, %rsp                     # Set up stack pointer
    call main                              # Execute main program
    halt                                   # Terminate program

# Sample linked list
    .align 8
ele1:
    .quad 0x00a
    .quad ele2
ele2:
    .quad 0x0b0
    .quad ele3
ele3:
    .quad 0xc00
    .quad 0

main:
    irmovq ele1,%rdi
    call rsum_list                           # rsum_list(ele1)
    ret

# long rsum_list(list *ls)
# ls in %rdi
rsum_list:
    pushq %r12
    irmovq $0,%rax
    andq %rdi,%rdi
    je test
    mrmovq (%rdi), %r12
    mrmovq 8(%rdi),%rdi
    call rsum_list
    addq %r12,%rax
    test: popq %r12
    ret

# Stack starts here and grows to lower addresses
    .pos 0x200
stack:
```

Listing 5: rsum.ys

1.3 copy.ys Copy a source block to a destination block

Solution The C version of copy_block() is on listing 6.

```
/* copy_block - Copy src to dest and return xor checksum of src */
long copy_block(long *src, long *dest, long len)
{
    long result = 0;
    while (len > 0) {
  long val = *src++;
  *dest++ = val;
  result ^= val;
  len--;
    }
    return result;
}
```

Listing 6: copy_block.c

Final version of copy.ys is on listing 7.

```
# Execution begins at address 0
    .pos 0
    irmovq stack, %rsp                        # Set up stack pointer
    call main                                 # Execute main program
    halt                                      # Terminate program

    .align 8
# Source block
src:
    .quad 0x00a
    .quad 0x0b0
    .quad 0xc00
# Destination block
dest:
    .quad 0x111
    .quad 0x222
    .quad 0x33

main:
    irmovq $3,%rdx
    irmovq dest,%rsi
    irmovq src,%rdi
    call copy_block                           # copy_block(src, dest, 3)
    ret

# long copy_block(long *src, long *dest, long len)
copy_block:
    irmovq $0,%rax
    andq %rdx,%rdx
    jle test
    loop: mrmovq (%rdi),%r10
    irmovq $8,%r11
    addq %r11,%rdi
    rmmovq %r10,(%rsi)
    addq %r11,%rsi
    xorq %r10,%rax
    irmovq $1,%r11
    subq %r11,%rdx
    andq %rdx,%rdx
    jg loop
    test: ret

# Stack starts here and grows to lower addresses
    .pos 0x200
stack:
```

Listing 7: copy.ys

2 Part B

Solution The updated hcl description of SEQ control signals for implementing IADDQ instruction show on listing 8.

```
################ Fetch Stage     ###################################

# Determine instruction code
word icode = [
  imem_error: INOP;
  1: imem_icode;    # Default: get from instruction memory
];

# Determine instruction function
word ifun = [
  imem_error: FNONE;
  1: imem_ifun;   # Default: get from instruction memory
];

bool instr_valid = icode in
  { INOP, IHALT, IRRMOVQ, IIRMOVQ, IRMMOVQ, IMRMOVQ,
         IOPQ, IJXX, ICALL, IRET, IPUSHQ, IPOPQ, IIADDQ };  ###### Add IIADDQ icode to inst_valid

# Does fetched instruction require a regid byte?
bool need_regids =
  icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ,
         IIRMOVQ, IRMMOVQ, IMRMOVQ, IIADDQ }; ###### Add IIADDQ icode to need_regids

# Does fetched instruction require a constant word?
bool need_valC =
  icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IJXX, ICALL, IIADDQ };  ###### Add IIADDQ icode to need_valC

################ Decode Stage    ###################################

## What register should be used as the A source?
word srcA = [
  icode in { IRRMOVQ, IRMMOVQ, IOPQ, IPUSHQ  } : rA;
  icode in { IPOPQ, IRET } : RRSP;
  1 : RNONE; # Don't need register
];

## What register should be used as the B source?
word srcB = [
  icode in { IOPQ, IRMMOVQ, IMRMOVQ, IIADDQ  } : rB;    ###### Add IIADDQ icode to srcB
  icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
  1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
word dstE = [
  icode in { IRRMOVQ } && Cnd : rB;
  icode in { IIRMOVQ, IOPQ, IIADDQ} : rB;    ###### Add IIADDQ to destE
  icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
  1 : RNONE;  # Don't write any register
];

## What register should be used as the M destination?
word dstM = [
  icode in { IMRMOVQ, IPOPQ } : rA;
  1 : RNONE;  # Don't write any register
];

################ Execute Stage   ###################################

## Select input A to ALU
word aluA = [
  icode in { IRRMOVQ, IOPQ } : valA;
  icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IIADDQ } : valC;    ###### Add IIADDQ to aluA
  icode in { ICALL, IPUSHQ } : -8;
  icode in { IRET, IPOPQ } : 8;
  # Other instructions don't need ALU
];

## Select input B to ALU
word aluB = [
  icode in { IRMMOVQ, IMRMOVQ, IOPQ, ICALL,
          IPUSHQ, IRET, IPOPQ, IIADDQ } : valB;    ###### Add IIADDQ to aluB
  icode in { IRRMOVQ, IIRMOVQ } : 0;
  # Other instructions don't need ALU
];

## Set the ALU function
word alufun = [
  icode == IOPQ : ifun;
  1 : ALUADD;
];

## Should the condition codes be updated?
bool set_cc = icode in { IOPQ, IIADDQ };    ###### Add IIADDQ to set_cc

################ Memory Stage    ###################################

## Set read control signal
bool mem_read = icode in { IMRMOVQ, IPOPQ, IRET };

## Set write control signal
bool mem_write = icode in { IRMMOVQ, IPUSHQ, ICALL };

## Select memory address
word mem_addr = [
  icode in { IRMMOVQ, IPUSHQ, ICALL, IMRMOVQ } : valE;
  icode in { IPOPQ, IRET } : valA;
  # Other instructions don't need address
];

## Select memory input data
word mem_data = [
  # Value from register
  icode in { IRMMOVQ, IPUSHQ } : valA;
  # Return PC
  icode == ICALL : valP;
  # Default: Don't write anything
];

## Determine instruction status
word Stat = [
  imem_error || dmem_error : SADR;
  !instr_valid: SINS;
  icode == IHALT : SHLT;
  1 : SAOK;
];

################ Program Counter Update ############################

## What address should instruction be fetched at

word new_pc = [
  # Call.  Use instruction constant
  icode == ICALL : valC;
  # Taken branch.  Use instruction constant
  icode == IJXX && Cnd : valC;
  # Completion of RET instruction.  Use value from stack
  icode == IRET : valM;
  # Default: Use incremented PC
  1 : valP;
];
#/* $end seq-all-hcl */
```

Listing 8: Update SEQ contorl signals for IADDQ instuction

3 Part C

Solution The optimized version of function ncopy() is shown on 9. Average CPE after optimization is **7.98**.

```
#/* $begin ncopy-ys */
##################################################################
# ncopy.ys - Copy a src block of len words to dst.
# Return the number of positive words (>0) contained in src.
#
# Include your name and ID here.
#
# Describe how and why you modified the baseline code.
#
##################################################################
# Do not modify this portion
# Function prologue.
# %rdi = src, %rsi = dst, %rdx = len
ncopy:

##################################################################
# You can modify this portion
  # Loop header
  xorq %rax, %rax
  iaddq $-5, %rdx
  jl EndUn

LoopUn:
  mrmovq (%rdi), %r10
  mrmovq 8(%rdi), %r11
  mrmovq 16(%rdi), %r12
  mrmovq 24(%rdi), %r13
  mrmovq 32(%rdi), %r9
  rmmovq %r10, (%rsi)
  rmmovq %r11, 8(%rsi)
  rmmovq %r12, 16(%rsi)
  rmmovq %r13, 24(%rsi)
  rmmovq %r9, 32(%rsi)

  andq %r10, %r10
  jle First
  iaddq $1, %rax
First:
  andq %r11, %r11
  jle Second
  iaddq $1, %rax
Second:
  andq %r12, %r12
  jle Third
  iaddq $1, %rax
Third:
  andq %r13, %r13
  jle Fourth
  iaddq $1, %rax
Fourth:
  andq %r9, %r9
  jle Fifth
  iaddq $1, %rax
Fifth:

  iaddq $40, %rdi   # src++
  iaddq $40, %rsi   # dst++
  iaddq $-5, %rdx   # len--
  jge LoopUn

EndUn:
  iaddq $5, %rdx
  je Done
  iaddq $-1, %rdx
  je One
  iaddq $-1, %rdx
  je Two
  iaddq $-1, %rdx
  je Three

  mrmovq (%rdi), %r10
  mrmovq 8(%rdi), %r11
  mrmovq 16(%rdi), %r12
  mrmovq 24(%rdi), %r13
  rmmovq %r10, (%rsi)
  rmmovq %r11, 8(%rsi)
  rmmovq %r12, 16(%rsi)
  rmmovq %r13, 24(%rsi)

  andq %r10, %r10
  jle F1
  iaddq $1, %rax
F1:
  andq %r11, %r11
  jle F2
  iaddq $1, %rax
F2:
  andq %r12, %r12
  jle F3
  iaddq $1, %rax
F3:
  andq %r13, %r13
  jle Done
  iaddq $1, %rax
  jmp Done

Three:
  mrmovq (%rdi), %r10
  mrmovq 8(%rdi), %r11
  mrmovq 16(%rdi), %r12
  rmmovq %r10, (%rsi)
  rmmovq %r11, 8(%rsi)
  rmmovq %r12, 16(%rsi)

  andq %r10, %r10
  jle Th1
  iaddq $1, %rax
Th1:
  andq %r11, %r11
  jle Th2
  iaddq $1, %rax
Th2:
  andq %r12, %r12
  jle Done
  iaddq $1, %rax
  jmp Done
Two:
  mrmovq (%rdi), %r10
  mrmovq 8(%rdi), %r11
  rmmovq %r10, (%rsi)
  rmmovq %r11, 8(%rsi)
  andq %r10, %r10
  jle T1
  iaddq $1, %rax
T1:
  andq %r11, %r11
  jle Done
  iaddq $1, %rax
  jmp Done
One:
  mrmovq (%rdi), %r10
  rmmovq %r10, (%rsi)
  andq %r10, %r10
  jle Done
  iaddq $1, %rax
##################################################################
# Do not modify the following section of code
# Function epilogue.
Done:
  ret
##################################################################
# Keep the following label at the end of your function
End:
#/* $end ncopy-ys */
```

Listing 9: Optimized version of ncopy.asm