Computer Science and Engineering 331

Midterm Examination #1

Fall 2000

Name: _____Solutions_____

S.S.#: _____

1	41	
2	13	
3	18	
4	28	
Total	100	

Instructions:

This exam contains 4 questions. It is closed book and notes. Calculators *are* allowed. Do all of your work on the attached sheets.

You have 75 minutes to complete the exam. The exam totals 100 points; the points for each problem are indicated. The points have been assigned according to the formula 1 point = 1 1/3 exam minutes, so pace yourself accordingly. This exam counts for about 24% of your course grade.

(41 pts) 1. Short Answer

A. Give three advantages of programming in a higher level language (like C) over programming in assembler (like MIPS).

1 – Improves programmer productivity (fewer lines of code, easier to understand,…)

2 – Programs are machine independent - portability

3 – Programs are easier to debug and maintain

   Good compilers can produce very efficient code

B. Label the remaining two states and show the transitions (by inserting arrows) for the finite state machine defining a machine instruction cycle:
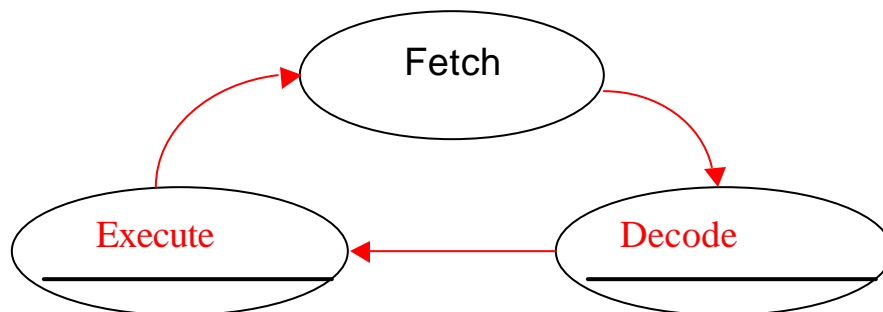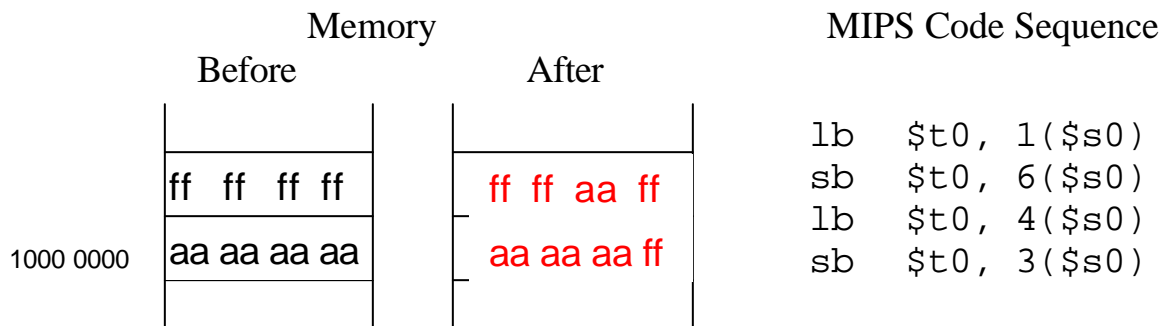

C. Give two reasons why registers are used as the source and destination operands for arithmetic instructions in the MIPS.

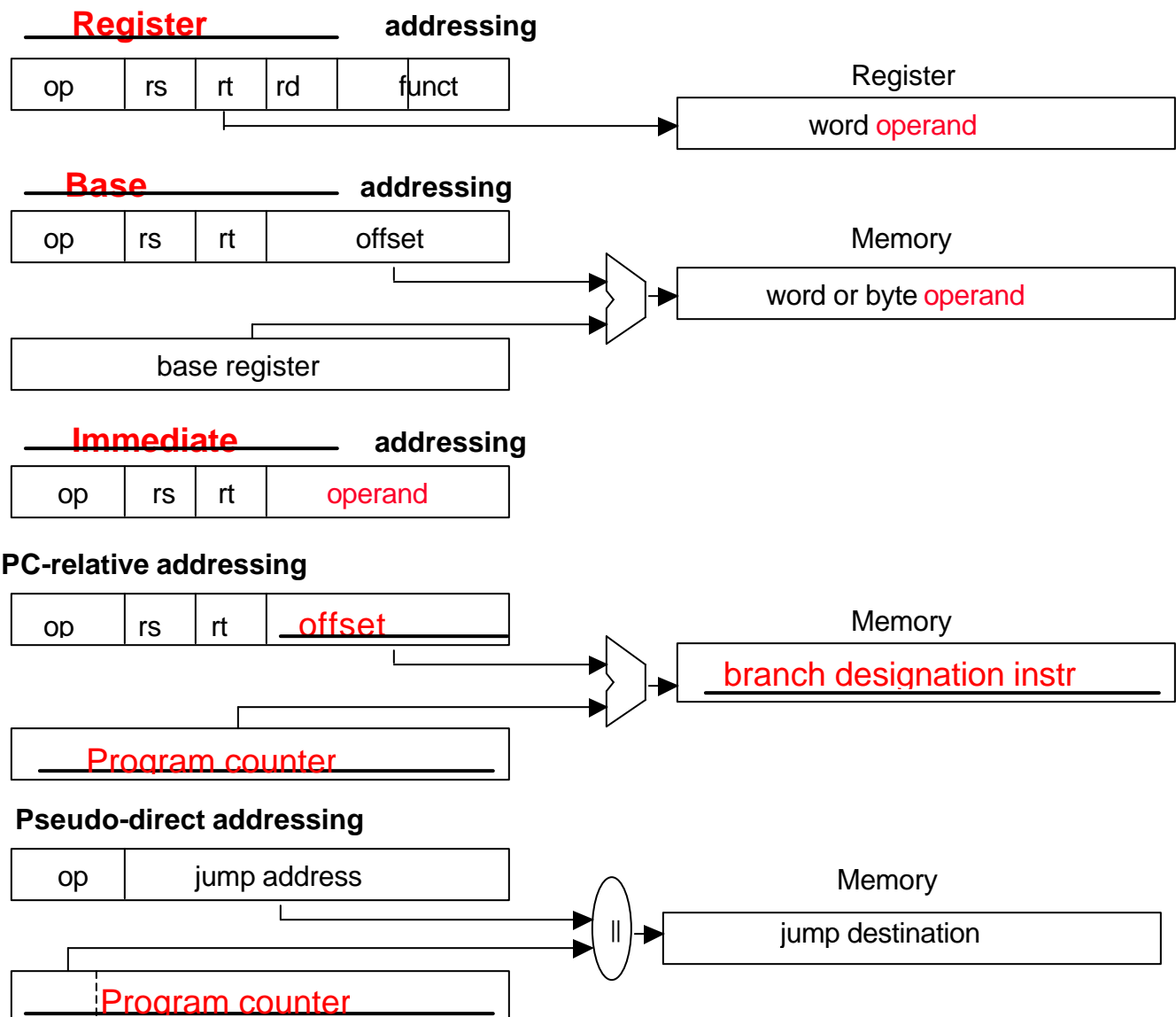1 – Faster than accessing operands from main memory

2 – Improved code density – takes fewer bits in the instruction to specify the operand addresses

   Easier for a compiler to use

D. Show the memory state (contents given in hexidecimal) after executing the following code sequence where $s0 is initialized to 0x1000 0000.

Memory		MIPS Code Sequence

Before   After

```
         ff  ff  ff  ff          ff  ff  aa  ff      lb   $t0, 1($s0)
                                                      sb   $t0, 6($s0)
1000 0000 aa aa aa aa            aa aa aa ff          lb   $t0, 4($s0)
                                                      sb   $t0, 3($s0)
```

E. Fill in the blanks in the diagram shown below for the MIPS addressing modes.

Register addressing

op	rs	rt	rd	funct

Register

word operand

Base addressing

op	rs	rt	offset

base register

Memory

word or byte operand

Immediate addressing

op	rs	rt	operand

PC-relative addressing

op	rs	rt	offset

Program counter

Memory

branch designation instr

Pseudo-direct addressing

op	jump address

Program counter

Memory

jump destination

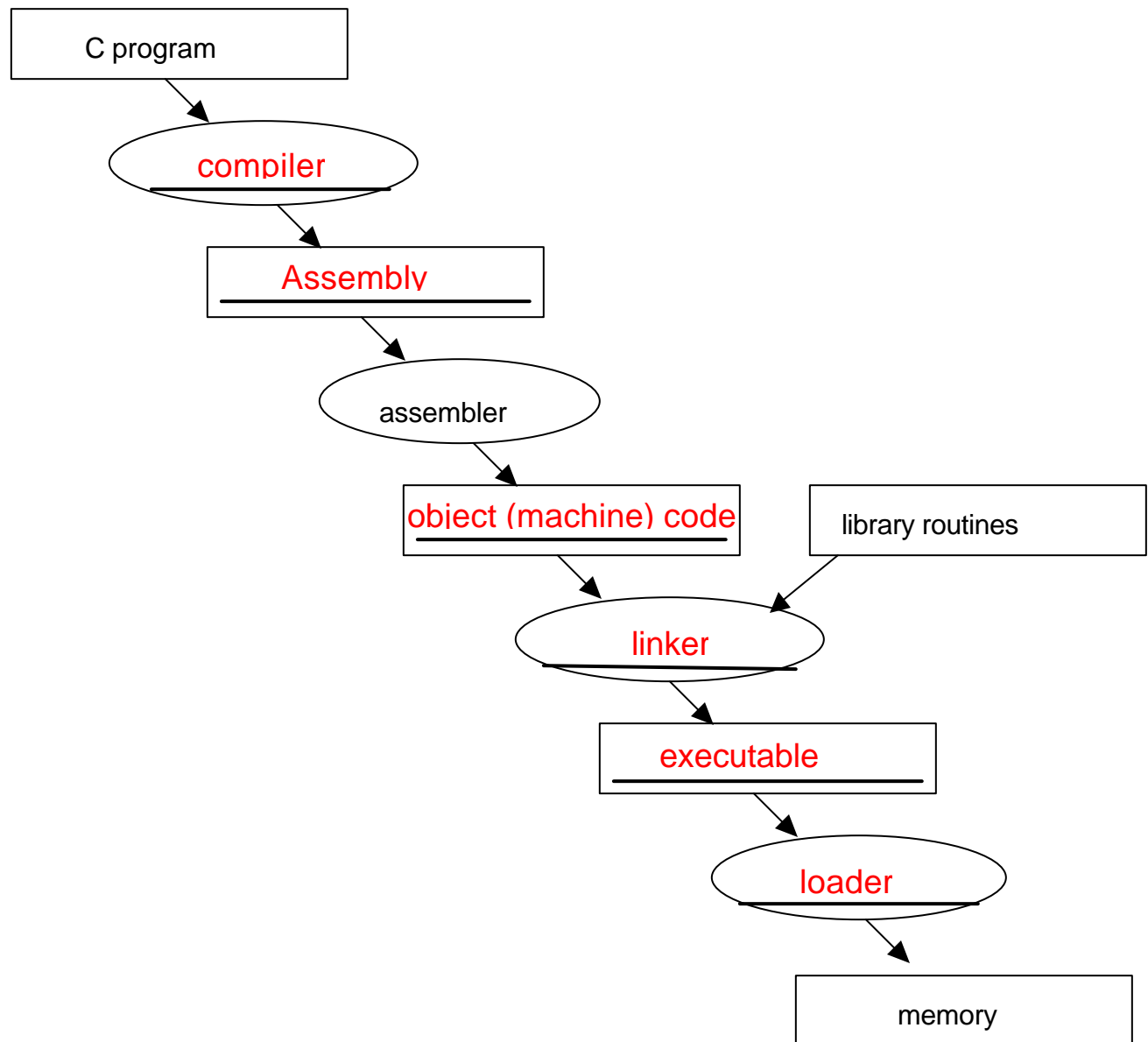F. Give the six steps in the execution of a procedure as discussed in class.

1 – Caller puts arguments where callee can find them (in $a0 - $a3 and/or on stack)

2 – Caller transfers control to the procedure (callee) – jal subroutine

3 – Callee acquires the storage resources needed (by spilling registers to the stack)

4 – Callee performs the desired task

5 –   Callee places result values where caller can find them (in $v0-$v1 and/or on stack)

6 – Callee returns control to the caller – jr $ra

G. List three tasks that the assembler performs (e.g., the SPIM assembler) in addition to its basic job of "assembling" the instructions (i.e., in addition to its job of filling in the instruction format with the binary opcode and register address information, computing branch offsets, and determining jump addresses).

1 – Converts pseudo-instructions to legal MIPS instructions

2 – Converts far away branches to a branch followed by a jump

3 – Converts instrs with large immediates into a lui followed by a ori

    Converts decimal values into binary and characters into their ASCII equivalents
    Builds the symbol table of instr. Labels and their addresses
H. Fill in the blanks in the translation hierarchy diagram shown below.

```
┌─────────────────┐
│    C program    │
└─────────────────┘
         ↓
    ⟨ compiler ⟩
         ↓
┌─────────────────┐
│    Assembly     │
└─────────────────┘
         ↓
    ⟨ assembler ⟩
         ↓
┌──────────────────────┐      ┌──────────────────┐
│ object (machine) code │      │ library routines │
└──────────────────────┘      └──────────────────┘
         ↓                            ↓
              ⟨ linker ⟩
                  ↓
         ┌─────────────────┐
         │   executable    │
         └─────────────────┘
                  ↓
              ⟨ loader ⟩
                  ↓
         ┌─────────────────┐
         │     memory      │
         └─────────────────┘
```

(13 pts) 2. Compiling MIPS Code

Only a very inefficient compiler would produce the code we derived in class for the C
while loop

C while loop	MIPS assembler

```
   while (i != k)          loop:     beq $s0, $s2, exit
       i = i + j                     add $s0, $s0, $s1
                                     j   loop
                           exit:     . . .
```

Give the MIPS assember code for the while loop that uses **at most** one branch or
one jump each time through the loop.

```
loop:   add $s0, $s0, $s1                    beq $s0, $s2, exit
        bne $s0, $s0, loop         loop:     add $s0, $s0, $s1
        sub $s0, $s0, $s1                    bne $s0, $s0, loop
        . . .                      exit:     . . .
```

```
        sub $s0, $s0, $s1                      j test
loop:   add $s0, $s0, $s1          loop:     add $s0, $s0, $s1
        bne $s0, $s0, loop         test:     bne $s0, $s0, loop
        . . .                                 . . .
```

How many instructions are executed if the loop is executed three times (e.g., if   i
= 0, j = 1, and k = 3 coming into the loop)?

Original code –   10

Your optimized code –   7 for all solutions except j test that takes 8

(18 pts) 3. Assembling MIPS Code

Give the machine code for the MIPS assembler code assembled at the starting address 0x0040 0000. (You may use a combination of decimal, hexidecimal and/or binary encodings.)

Address			MIPS assembler	
0x0040 0000		loop:	beq $s0, $s2, exit	
			add $s0, $s0, $s1	
			j loop	
		exit:	lui $s0, 48	

Machine Code:

bit   31

bit 0

beq	4	16	18	2

add	0	16	17	16	0	32

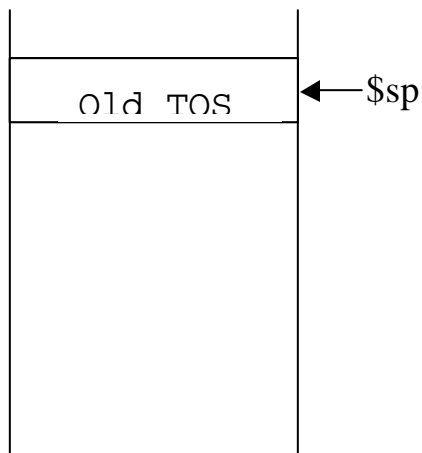| j	2	0x0100000

lui	15	0	16	48
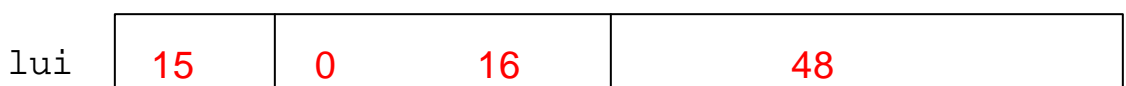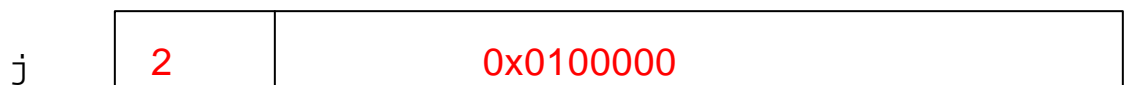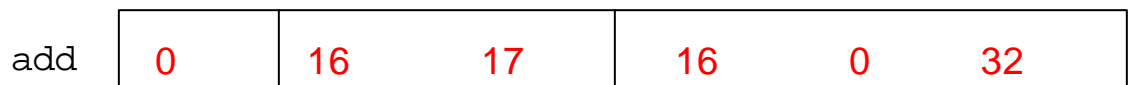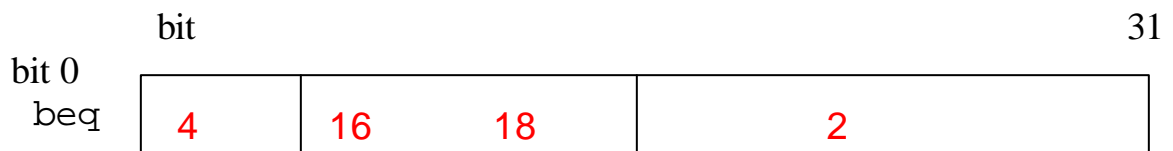
(28 pts) 4. Code Tracing and Subrouting Handling

For the following MIPS code sequence and given that $a0 = 1 initially, show the
contents of the program counter (PC), register $ra, and the memory stack after the
execution of each of the instructions marked below. Point 1 is given for you.

```
main:           . . .
            jal         sub1                        ←——— Point 1
next0:          . . .


sub1:       addi        $sp, $sp, -8
            sw          $ra, 4($sp)
            sw          $a0, 0($sp)
            bne         $a0, $zero, L1
            jr          $ra                         ←——— Point 3
L1:         addi        $a0, $a0, -1
            jal         sub1                        ←——— Point 2
next1:      lw          $a0, 0($sp)
            lw          $ra, 4($sp)
            addi        $sp, $sp, 8
            jr          $ra                         ←——— Point 4
```
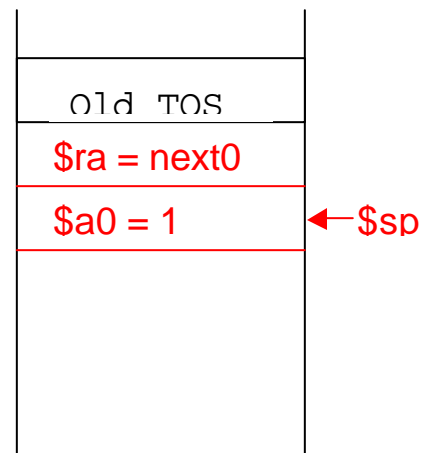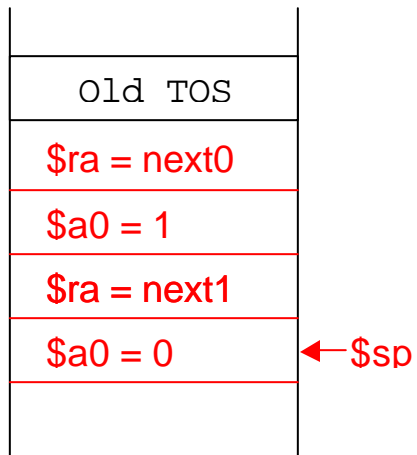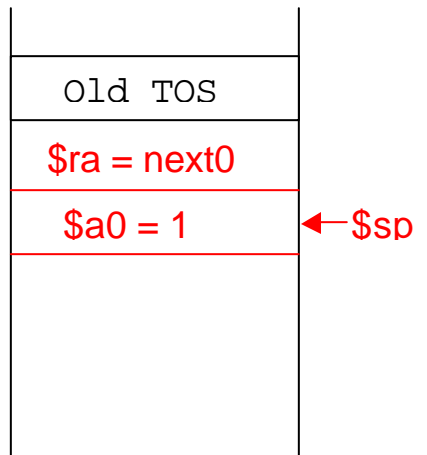
Point 1

Point 2

Point 3

Old TOS
$ra = next0
$a0 = 1
$ra = next1
$a0 = 0

←$sp

next1	$ra

next1	PC

Point 4 (first time)

Old TOS
$ra = next0
$a0 = 1

←$sp

next1	$ra

next1	PC

Point 4 (second time)

Old TOS

← $sp

next0	$ra

next0	PC