ECSE425 Computer Architecture and Organization

Assignment 3 Solutions

Linda Wang

Winter 2007

Question 1
==========

1. *a

```
        LW    T1, 0(Ra)
```

2. a <<= (b == c)

```
        BNE   Rb, Rc, L0  ;if b != c then do nothing to a
        DSLL  Ra, Ra, #1  ;otherwise left-shift a by 1
   L0:
```

3. a = expr1 || expr2

```
        DADDI Ra, R0, #1  ;initialize a to true
        BNEZ  E1, L0      ;if expr1 is true, a remains true
        BNEZ  E2, L0      ;or if expr2 is true, a remains true
        DADD  Ra, R0, R0  ;otherwise a is false
   L0:
```

4. a[expr] = a[expr-1] + b[expr+1]

```
        ;assume base addresses of a and b are in Ra and Rb,
        ;and assume a and b are integer arrays

        DSLL  T1, E1, #2  ;T1 = expr*4
        DADD  T2, Ra, T1  ;T2 = address of a[expr]
        DADD  T3, Rb, T1  ;T3 = address of b[expr]
        LW    T4, -4(T2)  ;T4 = a[expr-1]
        LW    T5, 4(T3)   ;T5 = b[expr+1]
        DADD  T4, T4, T5  ;T4 = a[expr-1] + b[expr+1]
        SW    T4, 0(T2)   ;store result to a[expr]
```

5. a -= (++a)

```
        DADDI Ra, Ra, #1  ;++a
        DSUB  Ra, Ra, Ra  ;a = a-a
```

6. a -= (a++)

```
        DSUB  Ra, Ra, Ra  ;a = a-a
        DADDI Ra, Ra, #1  ;a++
```

7. a & 0xAAAABBBBCCCCDDDD

```
        LUI   T1, #0xAAAA     ;T1 = 00000000AAAA0000
        ORI   T1, T1, #0xBBBB ;T1 = 00000000AAAABBBB
        DSLL  T1, T1, #32     ;T1 = AAAABBBB00000000
        LUI   T2, #0xCCCC     ;T2 = 00000000CCCC0000
        ORI   T2, T2, #0xDDDD ;T2 = 00000000CCCCDDDD
        OR    T1, T1, T2      ;T1 = AAAABBBBCCCCDDDD
        AND   T2, Ra, T1      ;T2 = a & AAAABBBBCCCCDDDD
```

8. do stmt while (expr)

```
   L0:  S1
        BEQZ  E1, L0
```

9. while (expr) stmt

```
   L0:  BEQZ  Eq, L1
        S1
        J     L0
   L1:
```

10. !a

```
        DADDI T1, T1, #1  ;initialize result to true
        BEQZ  Ra, L0      ;if a is false then change nothing
        DADDI T1, R0, R0  ;otherwise result is false
   L0:
```

Question 2

(a) Please see the end of this file for the assembly output of both the optimized and the optimized subroutines. Detailed comments were added to the assembly code.

(b) Discussion of unoptimized code:

The total number of memory transactions is estimated by counting the number of loads, the number of stores, and the number of all instructions per iteration in each subroutine. Then we multiply their sum by the number of iterations (100). The details of the multiplication subroutine 'umul' called is not known so its instructions are not counted.

Number of memory transactions in unoptimized code:

Subroutine	#Load/iter	#Store/iter	#Instr/iter	Total/iter	Total
foo1	8	2	44	54	5400
foo2	3	1	32	36	3600
foo3	3	1	21	25	2500

We see the following differences between the implementation of the three subroutines:

- foo1 uses global variables for everything: the two arrays, the array index, and glob_c. Reading a global variable requires calculating its address and loading it from memory. If the variable is changed, an additional store is needed. Therefore, the total number of memory transactions in foo1 is high.

- foo2 makes the array index a local register variable. This means that the index is always stored in a register, not in memory, so accessing it does not need loads, stores, and address calculations at all. As a result, foo2 only has 2/3 the memory transactions of foo1.

- foo3 goes further by making both arrays local as well. Now the arrays are stored on the stack, so array addresses can be calculated by simply adding an offset to the stack pointer. This further reduces the number of instructions (hence the number of memory transactions).

- As a note of caution, although foo3 has fewer memory transactions, it is generally NOT a good idea to make very large arrays local. This is because the stack space is limited and can easily overflow, resulting in program termination.

Since the code is not optimized, no instruction reordering has been done. The compiler could only put nops in the branch delay slots.

(c) Discussion of optimized code:

Number of memory transactions in optimized code:

Subroutine	#Load/iter	#Store/iter	#Instr/iter	Total/iter	Total
foo1	5	1	18	23	2300
foo2	3	3	15	21	2100
foo3	2	0	15	17	1700

Compared with unoptimized code, there is obviously a large reduction in the instruction count. A few examples of optimization are:

- The base address of the arrays and other variables are now calculated before the loop starts rather than being repeated in every iteration.

- The check for loop termination is moved to be AFTER the loop body, resulting in less branch instructions.

- Most delay slots are filled with useful instructions rather than nops.

- Instruction pairs that are data dependent are moved far apart in order to avoid stalls (for example, see how the sethi and or instructions used to calculate the address of glob_A are scheduled apart in optimized foo1).

Question 3

(a) Timing of code:

```
CC              1  2  3  4  5  6  7  8
--------------------------------------
DADD R1,R2,R3  IF ID EX ME WB
DADD R3,R4,R5     IF ID EX ME WB
DADD R5,R3,R4        IF ID EX ME WB
DADD R6,R7,R8           IF ID EX ME WB
```

At the end of the 5th cycle, R1 will be written, R7 & R8 will be read.

(b) In cycle 4, no useful data is forwarded. In cycle 5, the result of the execution stage in cycle 4 (i.e., the sum of R4 and R5) is forwarded to the input of the execution stage.

(c) Timing without forwarding hardware (assume BEQZ is not taken):

```
1 LD   R2,0(R3) IF ID EX ME WB
2 SLT  R1,R2,R0    IF -- -- ID EX ME WB
3 BEQZ R1,OUT                IF -- -- ID EX ME WB
4 NOP                                 IF ID EX ME WB
5 DSUB R2,R2,R0                          IF ID EX ME WB
6 SD   R2,0(R3)                             IF -- -- ID EX ME WB
7 DADD R3,R3,#8                                      IF ID EX ME WB
8 J    LOOP                                             IF ID EX ME WB
9 NOP                                                      IF ID EX ME WB
```

(d) Instructions 1 & 2: LD writes the value of R2 in its WB stage, but SLT needs R2 in its ID stage. Instructions 3 & 4: SLT writes the value of R1 in its WB stage, but BEQZ needs R1 in its ID stage. Instructions 5 & 6: DSUB writes the value of R2 in its WB stage, but SD needs R2 in its ID stage.

(e) Timing with forwarding hardware:

```
1 LD   R2,0(R3) IF ID EX ME WB
2 SLT  R1,R2,R0    IF ID -- EX ME WB
3 BEQZ R1,OUT         IF -- -- ID EX ME WB
4 NOP                          IF ID EX ME WB
5 DSUB R2,R2,R0                   IF ID EX ME WB
6 SD   R2,0(R3)                      IF ID EX ME WB
7 DADD R3,R3,#8                         IF ID EX ME WB
8 J    LOOP                                IF ID EX ME WB
9 NOP                                         IF ID EX ME WB
```

(f) Scheduled code:

```
LOOP: LD   R2, 0(R3)
      SLT  R1, R2, R0
      DADD R3, R3, #8
      BEQZ R1, OUT      ;delay slot
      DSUB R2, R2, R0
      J    LOOP
      SD   R2, -8(R3)   ;delay slot
OUT:
```

Timing of scheduled code:

```
LD   R2,0(R3) IF ID EX ME WB
SLT  R1,R2,R0    IF ID -- EX ME WB
DADD R3,R3,#8       IF -- ID EX ME WB
BEQZ R1,OUT               IF ID EX ME WB
DSUB R2,R2,R0                IF ID EX ME WB
J    LOOP                       IF ID EX ME WB
SD   R2,-8(R3)                     IF ID EX ME WB
```

(g) Two possible solutions (refer to Figure A.18 of text): 1. Add a path from the output of the MEM/WB register to the data input of data memory. 2. Add a path from the output of data memory to the input of the EX/MEM register. A multiplexer is needed in both cases to choose between the forwarded and the original values.

```
(h) Timing of code on the R4000 pipeline:

CC                 1  2  3  4  5  6  7  8  9 10 11
-------------------------------------------------
1 DADD R2,R4,R5  IF IS RF EX DF DS TC WB
2 DADD R4,R2,R5     IF IS RF EX DF DS TC WB
3 SD   R5,20(R2)       IF IS RF EX DF DS TC WB
4 DADD R3,R5,R4           IF IS RF EX DF DS TC WB

Forwarding done:

Instr pair | Forwarding
-----------+-----------
#1, #2     | EX to EX
#1, #3     | DF to EX
#2, #4     | DF to EX


Question 4
==========

(a) Timing of first iteration:

CC                1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
----------------------------------------------------------------------------------
L.D    F0,0(R2) IF ID EX ME WB
L.D    F4,0(R3)    IF ID EX ME WB
MUL.D  F0,F0,F4       IF ID -- M1 M2 M3 M4 M5 M6 M7 ME WB
SUB.D  F2,F0,F2          IF -- ID -- -- -- -- -- -- A1 A2 A3 A4 ME WB
DADDI  R2,R2,#8                IF -- -- -- -- -- -- ID EX ME WB
DADDI  R3,R3,#8                                     IF ID EX ME WB
DSUBU  R5,R4,R2                                        IF ID EX -- ME WB
BNEZ   R5,LOOP                                            IF -- -- ID EX ME WB
NOP                                                                IF ID EX ME WB

*Note that in cycle 17, DSUBU R5,R4,R2 is stalled for one cycle
before its MEM stage. This is done so that DSUBU won't enter MEM
at the same time as SUB.D F2,F0,F2.


(b) Scheduled code (there's more than one way of doing this):

LOOP: L.D    F0,0(R2)
      L.D    F4,0(R3)
      MUL.D  F0,F0,F4
      DADDI  R2,R2,#8
      DSUBU  R5,R4,R2
      SUB.D  F2,F0,F2
      BNEZ   R5,LOOP
      DADDI  R3,R3,#8

Timing of scheduled code:

CC                1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
----------------------------------------------------------------------
L.D    F0,0(R2) IF ID EX ME WB
L.D    F4,0(R3)    IF ID EX ME WB
DADDI  R2,R2,#8       IF ID EX ME WB
MUL.D  F0,F0,F4          IF ID M1 M2 M3 M4 M5 M6 M7 ME WB
DSUBU  R5,R4,R2             IF ID EX ME WB
SUB.D  F2,F0,F2                IF ID -- -- -- -- -- A1 A2 A3 A4 ME WB
BNEZ   R5,LOOP                    IF -- -- -- -- -- ID EX ME WB
DADDI  R3,R3,#8                                     IF ID EX ME WB
```

....

Assembly Output of Question 2
=============================

UNOPTIMIZED FOO1

```
foo1:
        !#PROLOGUE# 0
        save    %sp, -112, %sp
        !#PROLOGUE# 1
        nop

        ;; ---- glob_i = 0
        sethi   %hi(glob_i), %o1        ; get addr of glob_i
        or      %o1, %lo(glob_i), %o0
        st      %g0, [%o0]              ; glob_i = 0

        ;; ---- start of for loop (check if glob_i<100)
.LL3:
        sethi   %hi(glob_i), %o1        ; get addr of glob_i
        or      %o1, %lo(glob_i), %o0
        ld      [%o0], %o1              ; load glob_i
        cmp     %o1, 99                 ; compare glob_i with 99
        ble     .LL6                    ; if i<=99, go to loop body
        nop                             ; delay slot
        b       .LL4                    ; otherwise go to end of subroutine
        nop                             ; delay slot

        ;; ---- start of loop body
.LL6:
        sethi   %hi(glob_A), %o1        ; get base addr of glob_A
        or      %o1, %lo(glob_A), %o0
        sethi   %hi(glob_i), %o2        ; get addr of glob_i
        or      %o2, %lo(glob_i), %o1
        ld      [%o1], %o2              ; load glob_i
        mov     %o2, %o3
        sll     %o3, 2, %o1             ; glob_i*4 (for accessing glob_A[glob_i])
        sethi   %hi(glob_B), %o3        ; get base addr of glob_B
        or      %o3, %lo(glob_B), %o2
        sethi   %hi(glob_i), %o4        ; get addr of glob_i
        or      %o4, %lo(glob_i), %o3
        ld      [%o3], %o4              ; load glob_i
        add     %o4, 1, %o3             ; glob_i+1
        mov     %o3, %o4
        sll     %o4, 2, %o3             ; (glob_i+1)*4 (for accessing 
glob_B[glob_i+1])
        ld      [%o0+%o1], %f2          ; load glob_A[glob_i]
        ld      [%o2+%o3], %f3          ; load glob_B[glob_i]

        ;; ---- if (glob_A[glob_i] > glob_B[glob_i])
        fcmpes  %f2, %f3                ; compare glob_A[glob_i] and glob_B[glob_i]
        nop                             ; delay slot
        fbule   .LL5                    ; if glob_A[glob_i] <= glob_B[glob_i], go 
to loop end
        nop                             ; delay slot

        ;; ---- glob_c *= glob_i + 1
        sethi   %hi(glob_c), %o0        ; get addr of glob_c
        or      %o0, %lo(glob_c), %l0
        sethi   %hi(glob_c), %o1        ; get addr of glob_c
        or      %o1, %lo(glob_c), %o0
        sethi   %hi(glob_i), %o2        ; get addr of glob_i
        or      %o2, %lo(glob_i), %o1
        ld      [%o1], %o2              ; load glob_i
        add     %o2, 1, %o1             ; glob_i + 1
        ld      [%o0], %o0              ; load glob_c
        call    .umul, 0                ; call multiply subroutine
        nop                             ; delay slot
        st      %o0, [%l0]              ; store result
.LL7:
        ;; ---- end of loop: ++glob_i
.LL5:
        sethi   %hi(glob_i), %o1        ; get addr of glob_i
        or      %o1, %lo(glob_i), %o0
        sethi   %hi(glob_i), %o1        ; get addr of glob_i
        or      %o1, %lo(glob_i), %o0
        sethi   %hi(glob_i), %o2        ; get addr of glob_i
        or      %o2, %lo(glob_i), %o1
        ld      [%o1], %o2              ; load glob_i
        add     %o2, 1, %o1             ; glob_i + 1
        st      %o1, [%o0]              ; store result
        b       .LL3                    ; go back to start of loop
        nop                             ; delay slot

        ;; ---- end of subroutine (return glob_c)
.LL4:
        sethi   %hi(glob_c), %o1
        or      %o1, %lo(glob_c), %o0
        ld      [%o0], %o1
        mov     %o1, %i0
        b       .LL2
        nop
.LL2:
        ret
        restore
```

UNOPTIMIZED FOO2

```
foo2:
        !#PROLOGUE# 0
        save    %sp, -112, %sp
        !#PROLOGUE# 1

        ;; ---- rloc_i = 0
        mov     0, %l0                  ; note rloc is in register %l0

        ;; ---- start of for loop (check if rloc_i<100)
.LL9:
        cmp     %l0, 99                 ; compare rloc_i with 99
        ble     .LL12                   ; if rloc_i<=99 go to loop body
        nop                             ; delay slot
        b       .LL10                   ; otherwise go to end of subroutine
        nop                             ; delay slot

        ;; ---- start of loop body
.LL12:
        sethi   %hi(glob_A), %o1        ; get base addr of glob_A
        or      %o1, %lo(glob_A), %o0
        mov     %l0, %o2
        sll     %o2, 2, %o1             ; rloc_i*4
        sethi   %hi(glob_B), %o3        ; get base addr of glob_B
        or      %o3, %lo(glob_B), %o2
        add     %l0, 1, %o3             ; rloc_i + 1
        mov     %o3, %o4
        sll     %o4, 2, %o3             ; (rloc_i+1)*4
        ld      [%o0+%o1], %f2          ; load glob_A[rloc_i]
        ld      [%o2+%o3], %f3          ; load glob_B[rloc_i+1]

        ;; ---- if (glob_A[rloc_i] > glob_B[rloc_i+1])
        fcmpes  %f2, %f3                ; compare glob_A[rloc_i] with
glob_B[rloc_i+1]
        nop                             ; delay slot
        fbule   .LL11                   ; if glob_A[rloc_i] <= glob_B[rloc_i+1] go
to loop end
        nop                             ; delay slot

        ;; ---- glob_c *= rloc_i + 1
        sethi   %hi(glob_c), %o0        ; get addr of glob_c
        or      %o0, %lo(glob_c), %l1
        sethi   %hi(glob_c), %o1        ; get addr of glob_c
        or      %o1, %lo(glob_c), %o0
        add     %l0, 1, %o1             ; rloc_i + 1
        ld      [%o0], %o0              ; load glob_c
        call    .umul, 0                ; call multiplication subroutine
        nop                             ; delay slot
        st      %o0, [%l1]              ; store result
.LL13:
        ;; ---- end of loop: ++rloc_i
.LL11:
        add     %l0, 1, %l0
        b       .LL9                    ; go to start of loop
        nop

        ;; ---- end of subroutine (return glob_c)
.LL10:
        sethi   %hi(glob_c), %o1
        or      %o1, %lo(glob_c), %o0
        ld      [%o0], %o1
        mov     %o1, %i0
        b       .LL8
        nop
.LL8:
        ret
        restore


UNOPTIMIZED FOO3
----------------

foo3:
        !#PROLOGUE# 0
        save    %sp, -920, %sp
        !#PROLOGUE# 1

        ;; ---- loc_c = 0
        st      %g0, [%fp-820]

        ;; ---- rloc_i = 0
        mov     0, %l0

        ;; ---- start of for loop (check if rloc_i<100)
.LL15:
        cmp     %l0, 99                 ; compare rloc_i with 99
        ble     .LL18                   ; if rloc_i<=99, go to loop body
        nop                             ; delay slot
        b       .LL16                   ; otherwise go to end of subroutine
        nop                             ; delay slot

        ;; ---- start of loop body
```

```
.LL18:
        mov     %l0, %o1
        sll     %o1, 2, %o0             ; rloc_i*4
        add     %fp, -416, %o1          ; get addr of loc_A[rloc_i]
        add     %l0, 1, %o2
        mov     %o2, %o3
        sll     %o3, 2, %o2             ; (rloc_i+1)*4
        add     %fp, -816, %o3          ; get addr of loc_B[rloc_i+1]
        ld      [%o1+%o0], %f2          ; load loc_A[rloc_i]
        ld      [%o3+%o2], %f3          ; load loc_B[rloc_i+1]

        ;; ---- if (loc_A[rloc_i] > loc_B[rloc_i+1])
        fcmpes  %f2, %f3
        nop
        fbule   .LL17
        nop

        ;; ---- loc_c *= rloc_i + 1
        add     %l0, 1, %o1             ; rloc_i + 1
        ld      [%fp-820], %o0          ; load loc_c
        call    .umul, 0                ; call multiplication subroutine
        nop                             ; delay slot
        st      %o0, [%fp-820]          ; store result
.LL19:
        ;; ---- end of loop: ++rloc_i
.LL17:
        add     %l0, 1, %l0
        b       .LL15
        nop

        ;; ---- end of subroutine (return loc_c)
.LL16:
        ld      [%fp-820], %o0
        mov     %o0, %i0
        b       .LL14
        nop
.LL14:
        ret
        restore


OPTIMIZED FOO1
--------------

foo1:
        !#PROLOGUE# 0
        save    %sp, -112, %sp
        !#PROLOGUE# 1

        ;; ---- get addresses of glob_i, glob_c, glob_A, glob_B
        sethi   %hi(glob_i), %i0
        sethi   %hi(glob_c), %l3
        sethi   %hi(glob_A), %o0
        sethi   %hi(glob_B), %o1
        or      %o0, %lo(glob_A), %l2
        or      %o1, %lo(glob_B), %l1
        mov     %l3, %l0

        ;; ---- glob_i = 0
        st      %g0, [%i0+%lo(glob_i)]

        ;; ---- start of for loop
.LL6:
        ld      [%i0+%lo(glob_i)], %o0  ; load glob_i
        add     %o0, 1, %o2
        sll     %o0, 2, %o0
        sll     %o2, 2, %o1
        ld      [%l2+%o0], %f3          ; load glob_A[glob_i]
        ld      [%l1+%o1], %f2          ; load glob_B[glob_i+1]

        ;; ---- if (glob_A[glob_i] > glob_B[glob_i+1])
        fcmpes  %f3, %f2
        nop                             ; delay slot
        fbule   .LL5

        ;; ---- glob_c *= i + 1
        ld      [%l0+%lo(glob_c)], %o0  ; delay slot
        call    .umul, 0
        mov     %o2, %o1                ; delay slot
        st      %o0, [%l0+%lo(glob_c)]

        ;; ---- end of for loop (++glob_i, check if glob_i < 100)
.LL5:
        ld      [%i0+%lo(glob_i)], %o0
        add     %o0, 1, %o0
        cmp     %o0, 99
        ble     .LL6

        ;; ---- end of subroutine (return glob_c)
        st      %o0, [%i0+%lo(glob_i)]  ; delay slot
        ld      [%l3+%lo(glob_c)], %i0
        ret
        restore


OPTIMIZED FOO2
--------------

foo2:
        !#PROLOGUE# 0
        save    %sp, -112, %sp
        !#PROLOGUE# 1

        ;; ---- get address of glob_A, glob_B, glob_C
        sethi   %hi(glob_A), %o0
        sethi   %hi(glob_c), %l3
        or      %o0, %lo(glob_A), %l2
        sethi   %hi(glob_B), %o1
        or      %o1, %lo(glob_B), %l1
        mov     %l3, %l0

        ;; ---- rloc_i = 0
        mov     0, %o0
        add     %o0, 1, %i0

        ;; ---- start of for loop
.LL16:
        sll     %o0, 2, %o0
        sll     %i0, 2, %o1
        ld      [%l2+%o0], %f3          ; load glob_A[rloc_i]
        ld      [%l1+%o1], %f2          ; load glob_B[rloc_i+1]

        ;; ---- if (glob_A[rloc_i] > glob_B[rloc_i+1])
        fcmpes  %f3, %f2
        nop                             ; delay slot
        fbule   .LL12

        ;; ---- glob_c *= i + 1
        ld      [%l0+%lo(glob_c)], %o0  ; delay slot
        call    .umul, 0
        mov     %i0, %o1                ; delay slot
        st      %o0, [%l0+%lo(glob_c)]

        ;; ---- end of for loop (++rloc_i, check if glob_i < 100)
.LL12:
        mov     %i0, %o0
        cmp     %o0, 99
        ble     .LL16
        add     %o0, 1, %i0             ; delay slot

        ;; ---- end of subroutine (return glob_c)
        ld      [%l3+%lo(glob_c)], %i0
        ret
        restore


OPTIMIZED FOO3
--------------

foo3:
        !#PROLOGUE# 0
        save    %sp, -912, %sp
        !#PROLOGUE# 1

        ;; ---- loc_c = 0
        mov     0, %i0

        ;; ---- rloc_i = 0
        mov     0, %o0

        ;; ---- get addr of loc_A and loc_B
        add     %fp, -416, %l2
        add     %fp, -816, %l1
        add     %o0, 1, %l0

        ;; ---- start of for loop
.LL24:
        sll     %o0, 2, %o0
        sll     %l0, 2, %o1
        ld      [%l2+%o0], %f3          ; load rloc_A[rloc_i]
        ld      [%l1+%o1], %f2          ; load rloc_B[rloc_i+1]

        ;; ---- if (glob_A[rloc_i] > glob_B[rloc_i+1])
        fcmpes  %f3, %f2
        nop
        fbule   .LL20

        ;; ---- glob_c *= i + 1
        mov     %i0, %o0                ; delay slot
        call    .umul, 0
        mov     %l0, %o1                ; delay slot
        mov     %o0, %i0

        ;; ---- end of for loop (++rloc_i, check if rloc_i < 100)
.LL20:
        mov     %l0, %o0
        cmp     %o0, 99
        ble     .LL24

        ;; ---- end of subroutine (return rloc_c)
        add     %o0, 1, %l0             ; delay slot
        ret
        restore
```