NAME: SOLUTIONS

NUMBER: ___________________________

EE457--SPRING 2001

Midterm #1

Closed books, closed notes
DURATION: 1 hour and 20 minutes

Q1: /6
Q2: /6
Q3: /4
Q4: /4
Q5: /20

TOTAL: /40

Scaled Total: /15
QUESTION 1 (10minutes) 6pts

In a machine M1 clocked at 100MHz it was observed that 20% of the computation time of the spec95 integer point benchmarks was spent in the subroutine Multiply(A,B,C) which multiplies integer A and B and returns the result in C. Furthermore, each invocation of Multiply takes 800 cycles to execute. To speed up the program it is proposed to introduce a new instruction MULT to improve the performance of the machine on spec95 benchmarks. Please answer the following questions, if you have enough data. If there is not enough data simply answer “not enough data”.

(2pts) How many times is the Multiply routine executed in the set of programs?
Answer: Not enough data

(2pts) An implementation of the MULT instruction is proposed for a new machine M2. MULT executes the multiplication in 40 cycles (which is an improvement over the 800 cycles needed in M1.) Besides the Multiplies, all other instructions which were not part of the multiply routine in M1 have the same CPI in M1 and M2. Because of the added complexity however, the clock rate of M2 is 80MHz? How much faster (or slower) is M2 over M1?
Answer:
Cycle_count(M2) = .8xCycle_count(M1)+.2xCycle_count(M1)/20= .81xCycle_count (M1)

TEX(M1)/TEX(M2) = [Cycle_count (M1)/100] / [Cycle_count(M2)/80]

= [Cycle_count (M1) / Cycle_count(M2)] x .8 = .8 / .81

----------------------------------------------------------------------------------------------------------------

(2pts) A faster hardware implementation of the MULT instruction is designed and simulated for a proposed machine M3, also clocked at 80MHz. A speedup of 10% over M1 is observed. Is this possible or is there a bug in the simulator? If it is possible, how many cycles does the MULT instruction take in this new machine? If it is not possible, why is this so?
Answer:
This is not possible. According to Amdahl’s law, Cycle_count(M2) can never be less than Cycle_count(M1) x .8. So no speedup is possible. The best hope is that M2 is as fast as M1.
Question 2 (10 minutes) 6 points

The bytes at address 1000H through 1003H are filled with the following values:

1000H: 20H
1001H: F4H
1002H: 00H
1003H: 0AH

Assuming two’s complement arithmetic and the little endian byte addressing convention, give the numerical value in decimal of the 32-bit word stored at address 1000H and of each 16-bit half word stored at addresses 1000H and 1002H.

1. Value of word @(1000H) =

2. Value of half-word@(1000H)=____________________

3. Value of half-word@(1002H)=____________________

Answer:

Little endian means that the least significant byte is at the lower address.

So, the word content is 0A00F420H =

= (10x16^6 + 15x16^3 + 4x16^2 + 2x16) = 32 + 1024 + 61440 + 167772160 = 167834656

The content of half word at address 1000H = F420 = 1111 0100 0010 0000 =

- (1011 1110 0000) = - (11x16^2 + 14x16) = 2816 + 224 = 3040

The content of half word at address 1002H = 0A00 = 10x16^2 = 2560
Question 3 (10 minutes) 4 pts

The SLT $6,$4,$5 instruction stores 1 in $6$ if ($4)<($5) in two’s complement, otherwise, it
stores 0 in $6$. The SLTU instruction is similar, except that the comparison in unsigned
number arithmetic.

Using the following data in $4$ and $5$ (in hexadecimal), what will be the content of ($6)
after executing SLT $6,$4,$5 and SLTU $6,$4,$5 Please answer 0 or 1 in each empty
cell of Table 1.

<table>
<thead>
<tr>
<th>($4$)</th>
<th>($5$)</th>
<th>SLT $6,$4,$5</th>
<th>SLTU $6,$4,$5</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0000000H</td>
<td>80000000H</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EFFFFFFH</td>
<td>00800000H</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F0000000H</td>
<td>F0000000H</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01000000H</td>
<td>00000011H</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
QUESTION 4 (10 minutes) 4pts

Consider the following MIPS assembly code program:

```
L1: SLT $3,$4,$14
    BEQ $4,$0,L2
    LW $14,0($3)
    LW $4,15($14)
    ADD $4,$4,$4
    BEQ $4,$0,L1
L2: ADD $4,$4,$4
```

The assembler must compute the offset in the BEQ instruction. Give the contents in binary of the offset field (16 bits) in the machine instruction for the first BEQ in the program.

offset\textsubscript{15:0} = | 0 \_ 0 \_ 0 \_ 0 \_ 0 \_ 0 \_ 0 \_ 0 \_ 0 \_ 0 \_ 1 \_ 0 \_ 0 \_ 0 |

\text{TargetPC} = \text{BranchPC} + 4 + \text{Offset x 4} \Rightarrow \text{Offset} = (\text{TargetPC} - \text{BranchPC} - 4) / 4

= 5 - 1 = 4
QUESTION 5. FSM Design (40 minutes) 20 points

The design of a simple finite state machine (FSM) is the topic of this problem.

You will design an FSM which accepts a string of bits as input and detects isolated 0’s in the input string. An isolated 0 is a 0 surrounded by two 1’s.

More formally, the input is a M-bit binary string, $b_0b_1b_2b_3......b_{M-1}$.

$b_i$ is an isolated 0 iff $b_i = 0$, $b_{i-1} = 1$, and $b_{i+1} = 1$. We also adopt the convention that $b_{-1} = 0$ and $b_M = 0$. So, for example, 01010101 has 3 isolated 0’s

Besides the binary string, the machine takes two additional inputs, START and END. START and END are always 0, except in the clock period directly preceding the input of the first bit of the string (when START =1), and in the clock period directly following the last bit of the string (when END =1). Between START =1 and END =1, one new bit of the sequence is input at every clock. Finally, the binary sequence can have any length, including zero length, but START and END can never be both high in the same clock.

The output of the FSM is called “Up”. It is always 0, except in the clock directly following and isolated 0. So, for example, if the input string is 01010101 Up will be 00010101. Up is used to count the number of isolated zeros in an Up/Down counter (you don’t have to worry about this part).

The structure of this machine is shown in Figure 1. You need to design the FSM. Clearly FSM must be a MEALY machine.

![Figure 1. FSM Structure](image)

To design this machine we will use 4 states: WAIT, S00, SX1, and S10. When the machine is in state S00, the last two bits were 0 and 0. When it is in state SX1, the last bit was 1. And when it is in state S10, the last two bits were 1 and 0. WAIT is a waiting state where the machine stays after receiving END=1 and while START=0.
1. Please complete the state transitions in Figure 2. Since your machine is a Mealy machine transitions between state should be labelled with the value of inputs (START, END and b_i) as well as output (Up). To simplify, we define the following input functions: B = (b AND END); and ~B = (b AND END). Please use them as needed.
2. We propose to implement this machine using the one-hot method and D-type Flip-flops. The four flip-flops are shown in Figure 3. On Figure 3, please draw the next state logic as well as the logic for the output function Up.