John Gangemi
u68714612
CDA 4205
10/16/2014

# HOMEWORK #5

## 4.3

### 4.3.1

The critical path is I-mem -> Registers -> Multiplexer -> ALU -> Data Memory -> Multiplexer.
Total latency before MUL = 400ps + 200ps + 30ps + 120ps + 350ps + 30ps = 1130 ps
Total latency after MUL = 1130ps + 300ps = 1430ps

### 4.3.2

5% fewer instructions -> 100 - 5 = 95% instructions executed
speedup = (1/0.95) * (1130ps / 1430ps) = 1.0526 * 0.7902 = 0.8318
Since the speedup is less than 1 the addition of a multiply instruction actually slows down execution pretty significantly.

### 4.3.3

base cost = 1000 + 2*30 + 3*10 + 100 + 200 + 2000 + 500 = 3890
new cost = 3890 + 600 = 4490
estimated cost factor = new cost / base cost = 4490/3890 = 1.15
cost / performance ratio = 1.15 / 0.8318 = 1.383
The cost is drastically more while the performance is drastically less.

## 4.4

### 4.4.1

When fetching instructions the process is defined by reading the PC for the instruction's address, locating the instruction (I-Mem), and adding 4 to the instruction's address. The process of locating the instruction and adding 4 to the instruction's address take place concurrently. Given the latency of the addition is less than the latency for the instruction memory than only the I-Mem latency is considered for the cycle time.

cycle time = I-Mem latency = 200ps

### 4.4.2

Critical path for a single instruction processor of unconditional branch type:

I-Mem -> Sign Extend -> Shft Lft 2 -> Add -> MUX
cycle time = 200ps + 15ps + 10ps + 70ps + 20ps = 315ps

**4.4.3**

Critical path for a single instruction processor of conditional branch type:

I-Mem -> Regs -> MUX -> ALU -> MUX

cycle time = 200ps + 90ps + 20ps + 90ps + 20ps = 420ps

**4.4.4**

Branch instructions require the output of the 'shift left 2' datapath resource

**4.4.5**

Critical path for *unconditional branches*: I-Mem -> Sign Extend -> *Shft Lft 2* -> Add -> MUX

**4.4.6**

For BEQ and ADD instructions the 'shift left 2' resource is not part of the critical path with initial conditions.

Given the BEQ instruction is of type I format then the resource could become part of the critical path if the latency has increased to an amount greater than the latency of the 'register block' minus the latency of the 'sign extend block' thus the cycle time will increase.

Since the ADD instruction is of type R format the 'shift left 2' resource can not become part of the critical path and thus the cycle time remains unaffected.

## 4.5

**4.5.1**

Data memory is used for the load word (lw) and store word (sw) instructions:

25% + 10% = 35% of all cycles

**4.5.2**

Sign extension is needed for all I format instructions:

addi + beq + lw + sw = 20% + 25% + 25% + 10% = 80% of all cycles

When sign extension is not needed the add and not instructions are being performed.

## 4.7

Given the opcode from the instruction is 101011 it can be found that this instruction is in the R format and specifically "sltu" or set on less than unsigned.

opcode = 101011

rs = 00011

rt = 00010

rd = 00000

shamt = 00000

function = 010100

**4.7.1**

Output of sign extend block: 16 bit extension + rd + shamt + function +

0000 0000 0000 0000 0000 0000 0010 0100

Output of shift left 2 block: rs + rt + rd + shamt + function + 2 LSBs

0001 1000 1000 0000 0000 0101 00$\underline{00}$

**4.7.2**

Values of ALU control inputs: Instr[5-0] & ALU Op[1-0] = 010100 and 00

**4.7.3**

PC address is incremented by 4 since this is an R format instruction, therefore the branch address arithmetic logic unit is bypassed using a multiplexer that feeds back into the PC.

**4.7.4**

*Write register MUX*: Instr[15-11] or Instr[20-16] = 0 or 2

*ALU MUX*: sign extend block is not considered with R format instruction therefore output is given by read data 2 = 2

*Mem/ALU MUX*: rs section of instruction is not less than rt section and thus the ALU result is zero and said value is used to write to rd (value is sent through MUX)

*Branch MUX*: PC + 4

*Jump MUX*: PC + 4

**4.7.5**

ALU inputs: read data 1 and read data 2 = 3 and 2

ADD (next instruction): PC and constant 4

ADD (branch instruction): PC + 4 and (20 * 4)