Ryan Young

CSE 3666

11/4/19

Homework 5

1.5) Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3GHz close speed and a CPI of 1.5. P2 has a clock speed of 2.5GHz and a CPI of 1.0. P3 has a clock speed of 4.0 GHz and a CPI of 2.2.

a.) Which processor has the highest performance expresses in instructions per second?

Performance = (Clock Rate) / (CPI)

Performance(CPU1) = = 2 \* 108

Performance(CPU2) = = 2.5 \*108

Performance(CPU3) = = 1.8181818181818188

So, CPU2 has the highest performance of the three CPU’s.

b.) If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

Number of cycles = Time \* Clock Rate

Number of Instructions = Number of Cycles / CPI

Number of Cycles(CPU1) = 10s \* 3.0GHz = 30\*109

Number of Instructions(CPU1) = 30\*109 / 1.5 = 20\*109

Number of Cycles(CPU2) = 10s \* 2.5GHz = 25\*109

Number of Instructions(CPU2) = 35\*109 / 1.0 = 25\*109

Number of Cycles(CPU3) = 10s \* 4.0GHz = 40\*109

Number of Instructions(CPU3) = 40\*109 / 2.2 = 18.181818181818181818181818\*109

c.) We are trying to reduce the execution time by 30% but this leads to an increase in 20 percent in the CPI. What clock rate should we have to get this time reduction?

10 \* (1 – 0.3) = 7

CPU1: 1.2 \* CPI = 1.8

(20\*109\* 1.8) / 7 = 5.14 GHz

CPU2: 1.2 \* CPI = 1.2

(25\*109\* 1.2) / 7 = 4.28 GHz

CPU3: 1.2\* CPI = 2.64

(40\*109\* 2.64) / 7 = 6.85 GHz

1.6) Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2. Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which implementation is faster?

a.) What is the global CPI for each implementation?

CPU time = (# of instructions \* CPI ) / Clock Rate

CPU time(p1)

=1.0\*106 \* ((0.1 \* 1) +(0.2\*2)+(0.5\*3) + (0.2\*3))) / (2.5\*109)

=1.04 ms

Global CPI for P1 = CPI \* Clock Rate / # of instructions

(1.04\*10-3 \* 2.5\*109) / 106

= 2.6

CPU time(p2)

=1.0\*106 \* ((0.1 \* 2) +(0.2\*2)+(0.5\*2) + (0.2\*2))) / (3.0\*109)

= 0.67 ms

Global CPI for P2 = CPI \* Clock Rate / # of instructions

(0.67\*10-3 \* 3.0\*109) / 106

= 2.01

b.) Find the clock cycles required in both cases?

Clock Cycles = Global CPI \* # of instructions

P1:

2.6 \* 106 = 26\*105

P2:

2.01 \* 106 = 20.1\*105

1.7) Compliers can have a profound impact on the performance of an application. Assume that for a program, complier A results in a dynamic instruction count of 1 \* 109 and has an execution time of 1.1 s, while complier B results in a dynamic instruction count of 1.2 \* 109 and an execution time of 1.5s.

a.) Find the average CPI for each program given that the process has a clock cycle time of 1 ns.

Complier A:

CPI = (CPU execution time) / ( # of instructions \* Clock Cycle Time)

1.1/ (1\*109 \* 10-9)

Average CPI  for complier A = 1.1

Complier B:

1.5/ (11.2\*109 \* 10-9)

Average CPI  for complier B = 1.25

b.) Assume the complied programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running complier A’s code versus the clock of the processor running compiler B’s code?

Execution time = # of instructions \* average CPI \* Clock Cycle Time

The two processors have the same execution time.

So, (1.2\*109 \* 1.25)/(1.1\*109) = 1.36

Which means that the clock of the processor running complier A is 1.36 times the clock of the processor running complier B.

c.) A new complier is developed that uses only 6.0 \* 108 and has an average CPI of 1.1. What is the speedup of using this new complier versus using compiler A or B on the original processor?

Speedup = execution time of old complier / execution time of new complier

New Complier CPU execution time = 0.6\*109 \* 1.1 \* 10-9

New Complier CPU execution time = 0.66 s

Speedup Versus A : 1.1 / 0.66 = 1.67

Speedup Versus B: 1.5 / 0,66 = 2.27

4.4) Problems

4.4.1) If the only thing we need to do in a processor is fetch consecutive instructions what would the cycle time be?

According to the table the latency for Instruction memory is 200 ps and the add latency is 70 ps. And since we are doing consecutive fetch instructions the length of the clock would have to match the longest latency. Making the cycle time be 200ps.

4.4.2) Consider a datapath similar to the one in figure 4.11 but for a processor that only has one type of instruction: unconditioned PC-relative branch. What would the cycle time be for this datapath?

The critical path for an unconditional PC-relative branch for a datapath similar to that show in figure 4.11 would have to start with instruction memory fetch with a latency of 200 ps this is done at the same time as the add function that adds 4 to the pc and since the adder has a lower latency we would only have clock cycle of 200 ps. Moving on next we would have to sign extend the value taking a time of 15 ps and shift that value to the left by 2 places taking 10 more ps. Next, we would use the add function to add that shifted value and the pc + 4 for a latency of 70ps and finally the MUX would select which value to use the pc + 4 or this new value taking 20 ps. This results in the critical path taking a time of 200+15+10+70+20 = 315 ps.

4.4.3) Repeat 4.4.3, but this time we need to support only conditional PC-relative branches.

Since this time, we need to support only conditional PC-relative branches we will need to use the register block as well. So, starting out we would have to start with instruction memory fetch with a latency of 200 ps. Moving on next we would have to use the register block taking a time of 90 ps. Next, we would use the mux to decide between the register value and the sign extended value but, in this case, we want the register value, for a latency of 20ps. After the mux the next element used is the ALU which has a time taken of 90ps and finally we would use the last mux which adds a further 20 ps to the critical path. This results in the critical path taking a time of 200+90+20+90+20 = 420 ps.

4.4.4) What kinds of instructions require this resource?

The kinds of instructions that would be executed on this resource/data paths would be branch instructions, either unconditional or conditional PC-relative branches.

4.4.5) For which kinds of instructions (if any) is this resource on the critical path?

The kinds of instructions that this resource is on the critical path are PC-relative unconditional branch instructions. Conditional branches are not required on the critical path however in MIPS there are not unconditional branches, we only use functions such as “BNE”, “BEQ” or “BLT”.

4.4.6) Assuming that we only support beq and add instructions, discuss how changes in the given latency of this resource affect the cycle time of the processor. Assume that the latencies of other resources do not change.

The beq instruction takes much longer than the add function to complete, so any changes in the add path should not affect the total time unless they are drastic. Changes in the beq critical path however will change the time because that is the path driving the run time. If we change the given latencies of this resource, it will change the cycle time of the processor. For example, if we change the latency of the instruction memory then anytime that element in the path is used it is going to change the clock cycle because it has the highest latency.

4.5) Problems

4.5.1) In what fraction of all cycles is the data memory used?

Data memory is used in two of the six instructions in the table, load word and save word. So, if 25 percent of instructions used are load word instructions and 10 percent of instructions are load word instructions then 35 percent of instructions need the data memory data path element.

4.5.2) In what fraction of all cycles is the input of the sign-extend circuit needed? What is the circuit doing in the cycles in which its input is not needed?

The sign extend is not need when performing the add and the not instructions so, 20% + 25% + 25% + 10% gives us 80%.

4.7) Problems

4.7.1) What are the outputs of the sign-extend and the jump “shift left 2” unit for this instruction word?

The OP code for the instruction is: 101011

The instruction word is bits 0-15: 000000000010100. Since the sign bit is a 0 the sign extending will take place with 0’s rather than 1’s.

So, sign extending the instruction word would result in 00000000000000000000000000010100.

The shift left 2 takes in the instruction [25-0] and outputs 28 bits. So, this would take in: 00011000100000000000010100 and then output 0001100010000000000001010000.

4.7.2) What are the values of the ALU control unit’s for this instruction?

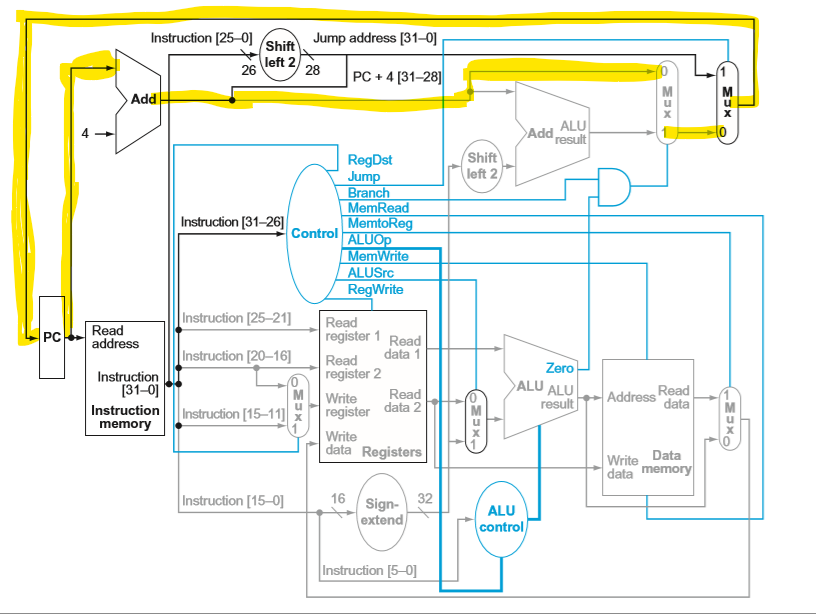
The function word is 010100 or the last 6 bits from the initial instruction.

Since we know that the ALUOp is taken from the two least significant bits from the function word. For the save word instruction ALUOp is 00.

4.7.3) What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

The new value of PC would be the current PC plus 4.

Below I highlighted the path through which the next pc value is calculated. First it goes from the current pc to the adder where 4 is added to the pc. Then it moves through the branch mux and the jump mux and comes back to the pc. Obviously, these mux do not do anything in the process of calculating the new pc. However, if we wanted to jump to the new pc as well then, they would need to get the control inputs and a new path would need to be highlighted.



4.7.4) For each Mux, show the values of its data output during the execution of this instruction and these register values.

Since this is a store word operation not a branch operation the mux selecting pc + 4 or a branch address will be PC + 4. For a save word instruction RegDest is don’t care meaning that it can be a one or a zero . This is because we do not care which register, we are writing data too because we are not writing data to any registers. The mux right before the ALU must select the sign extended version of the 0-15 bits of the instruction. This means that the output of that mux must be 2010.

4.7.5) For the ALU and the two add units, what are their data input values?

The ALU will take in the read data that we are going to store in memory and the sign extended version of 0 – 15 of the instruction (-3). The add that operates on the PC (the left one) will take in PC and 4, while the right one will take in PC+4 and the sign extended version of 0-15 instructions shifted left twice (80).

4.7.6) What are the values of all inputs for the “register” unit?

The value for register 1 is instruction[25-12] (00011) and the value for register 2 is instruction[20-16] (00010). Since this is a save word instruction the value for RegDST is don’t care so it can be either a 0 or 1. And the value for RegWrite is 0 because again it is the save word, so we do not want to write to registers. This make the input for write data invalid because we have said we don’t want to write to any registers.

7.)

7.1) What are the values of control signals generated by the control in figure 4.11 for this instruction?

The values for the control signals generated by the control for this instruction are:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegWrite | MemRead | | MemWrite | ALU Op | | ALUSrc | | PCSrc | MemtoReg |
| a. | 1 | | 0 | 0 | 1 | 0 | | 0 | | 0 |
| b. | X | | 0 | 1 | 0 | 1 | | 0 | | X |

7.2) Which resources perform a useful function for this instruction?

In this instruction the Instruction memory is used, the Mux before the registers block, the registers block the Mux before the ALU and the ALU. The PC is also set to PC+4 automatically so in that case then the two top add blocks are used along with the Mux.

7.3 Which resources produce outputs, but their outputs are not used for this instruction? Which resources produce no outputs for this instruction?

a. The resource that produces an output but it does not get used is the branch path (sign extend, shift left 2, and the add ALU), while the resource that does not produce an output is the data memory block.

b. The resource that produces an output but it does not get used is the branch path (sign extend, shift left 2, and the add ALU and I don’t believe that any elements have no output.

8.) Different execution units and blocks of digital logic have different latencies (time needed to do their work.) In Figure 4.17 there are seven kinds of major blocks. Latencies of blocks along the critical (longest-latency) path for an instruction determine the minimum latency of that instruction. For the remaining three problems in this exercise, assume the following latencies for implementations a and b (12 points) :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | I-Mem | Add | Mux | ALU | Regs | D-Mem | Control |
| a. | 200 ps | | 70 ps | 20 ps | 90 ps | 90ps | 250 ps | 40 ps |
| b. | 750 ps | | 200 ps | 50 ps | 250 ps | 300ps | 500 ps | 300 ps |

8.1) What is the critical path for a MIPS AND instruction in both implementations (4 points)?

The critical path would be I-Mem – Control – Regs – Mux – ALU – Mux – Regs.

1. 200 + 40 + 90 + 20 + 90 + 20+ 90= 550 ps
2. 750 + 300 + 300 + 50 + 250 + 50 + 300 = 1630 ps

8.2) What is the critical path for a MIPS load (LD) instruction in both implementations (4 points)?

The critical path for a MIPS load instruction would be I-Mem – Control - Regs– Mux – ALU – D-Mem – Mux – Regs.

1. 200 + 40 + 90 + 20 + 90 + 250 + 20 + 90 = 800
2. 750 + 300 + 300 + 50 + 250 + 500 + 50 + 300 = 2500

8.3) What is the critical path for a MIPS BEQ instruction in both implementations (4 points)?

The critical path for MIPS BEQ instruction in both implementations: I-Mem – Control – Regs – Mux – Add – Mux.

1. 200 + 40 + 90 + 20 + 70 + 20 = 440
2. 750 + 300 + 300 + 50 + 200 + 50 = 1650