Computer Science 605.611

Problem Set 1 Answers

1. (4) The average CPI (cycles per instruction) for a certain program is 1.6. If the native MIPS rating for the program is 12.5, what is the corresponding CPU clock cycle time (i.e., the clock period) in units of nano-seconds (i.e., 10^-9 seconds)?

The Native MIPS = clock_rate / (10^6 * CPI_{avg})

So clock_rate = Native MIPS * (10^6 * CPI_{avg}) = 12.5 *(10^6 * 1.6) = 20 MHz.

Cycle time = 1/clock_rate = 1/20MHz = 0.5 * 10^-7 seconds = 50 nano-seconds.

2. Assume that MIPS instructions are processed in only 2 phases (a fetch phase and an execute phase). Each phase takes one clock cycle and, when possible, the fetch phase of one instruction is overlapped with the execute phase of another instruction. Consider the following instruction sequence:

```
lui     $2,0x4000       ; put the address 0x40000000 into register $2
lh      $8,4($2)        ; load the contents of a memory halfword into $8
lw      $9,8($2)        ; load the contents of a memory word into $9
add     $9,$9,$8        ; compute the sum of the two words
sw      $9,12($2)       ; store the sum into a third memory word
```

What is the total number of clock cycles required to execute this sequence of five instructions on:

a) (4) a MIPS system with a von Neumann architecture?

A von Neumann has one memory that contains both instructions and memory data operands. Hence the execution of a memory access instruction (such as lw or sw) can't be overlapped with the fetch of any other instruction. So the instructions are processed as follows:

cycle	action
1	fetch lui
2	fetch lh and execute lui
3	execute lh $8
4	fetch lw
5	execute lw
6	fetch add
7	execute add and fetch sw
8	execute sw

So 8 clock cycles are needed.

b) (4) a MIPS system with a Harvard architecture?
The Harvard architecture includes separate instruction and data memories, each with a separate CPU to memory bus. This allows the fetch of any instruction to be overlapped with the execution of another instruction.
So the instructions are processed as follows:

cycle	action
1	fetch lui
2	fetch lh and execute lui
3	execute lh $8 and fetch lw
4	execute lw and fetch add
5	execute add and fetch sw
6	execute sw

Only 6 cycles are required to complete the entire group of the instructions.

3. (4) Which is more likely to adhere to a load/store architecture: a RISC processor or a CISC processor? Explain your answer.
Since the emphasis is on maximum speed with RISC processors and instructions that employ memory operands take longer to execute due to the time it takes to access the data memory, a RISC processor is more likely to adhere to a load/store architecture so that access to the data memory is restricted to just the load/store type instructions. This also makes the common case fast since most RISC instructions do not reference memory operands. The emphasis with CISC processors is more on flexibility, so they are more likely to use arithmetic and logic instructions that employ memory data operands.

4. A certain program runs on a MIPS processor executing a total of 7 million instructions. The CPU clock rate is 4 GHz and the average CPI for the program is 2.5.

a) (3) What is the CPU clock cycle time?
Cycle_time = 1/clock_rate = 1/(4 GHz) = 1/ $(4 * 10^{9}) = 0.25 * 10^{-9}$ or 0.25 nano-seconds

b) (4) What is the execution time required for the program? Express your answer in micro-seconds.
Execution time = IC*CPI/clock_rate = $(7*10^{6} * 2.5)/4*10^{9}$ = 4.375*10-3 seconds = 4375 micro-seconds.

c) (4) What is the numeric value for the expression: clock_rate / $(CPI*10^{6})$ for this program?
Clock_rate/$(CPI*10^{6}) = 4*10^{9}/(2.5*10^{6})$ = 4000/2.5 = 1600, which is the MIPS rating.

d) (4) Evaluate the expression IC / (execution_time$*10^{6}$) for this program (where IC is the instruction count).
$7*10^{6}/(4375*10^{-6}*10^{6}) = 7*10^{6}/4375 = 1600$

5. (5) The table below shows the number of clock cycles required to execute various types of instructions on a version of our MIPS processor:

Instruction type	Cycles consumed by the instruction
R-type	4
lw	5
sw	4
beq	3
j	3

A certain program that executes on this version of the processor executes 782000 R-type instructions, 43506 lw instructions, 36897 sw instructions, 21300 beq instructions and 7200 j (jump) instructions.
If the native MIPS rating for the program is 448, what is the corresponding CPU clock rate?
Cycles consumed = 4*782000 + 5*43506 + 4*36897 +3*21300 + 3*7200 = 3578618
CPI_{avg} = 3578618/890903 = 4.017
The Native MIPS = clock_rate / (10^6 * CPI_{avg})
So clock_rate = Native MIPS * (10^6 * CPI_{avg}) = 448 * 4.017 * 10^6 = 1799.6 * 10^6 = 1.7996 GHz

6. (10) A program that executes forty million instructions runs on a uni-processor system with a fixed clock rate of 3.333 GHz. All instructions are executed one at a time and each instruction requires an integral (i.e., whole) number of clock cycles. The average CPI rating for the instructions is 3 cycles per instruction. What is the speedup for the program obtained by reducing the average CPI from 3 down to 2?
Answer: The 3.333 GHz clock rate corresponds to a cycle time of 1/3.333GHz = 0.3 nano-seconds. If the average CPI is 3, then the execution time for the program is 3*4*10^7 * 0.3 nano-seconds = 3.6*10^7 nano-seconds. Reducing the average CPI by 1 taking it from 3 to 2 would yield an execution time of 2*4*10^7 * 0.3 nano-seconds = 2. 4*10^7 nano-seconds. Hence the speedup is 3.6/2.4 = 1.5. Speedup is the ratio of the execution time before the improvement divided by the execution time after the improvement.

7. A number of instructions from five different instruction classes are executed for a program that runs on a certain processor. The five classes along with the number of instructions executed from each class and the CPI for each class are shown in the table below:

Class	Instruction type	CPI	Millions of instructions executed
A	R-type	8	20
B	lw	12	18
C	sw	10	12
D	beq	6	10
E	j	4	6

a) (5) What is the average CPI for the program?
The total number of cycles (in millions) consumed
= 8*20 + 12*18 + 10*12 + 6*10 + 4*6 = 580
The total number of instructions executed = 20+18+12+10+6 = 66 million
So the average CPI = 580/66 = 8.79

b) (10) If each beq instruction takes 12 nano-seconds to execute, what is the total execution time for the program?
Since the beq instruction takes 6 cycles, the CPU clock period = 12ns/6 = 2ns
The execution time = number of cycles consumed * clock period (i.e., cycle time)
Execution time = 580 million * 2 ns = $580*10^6 * 2 * 10^{-9} = 1160 * 10^{-3}$ seconds
= 1.16 seconds.

c) (5) Assume that you have the option of improving only one of the instruction classes. If the improvement is by a factor of 2, which class should be chosen for improvement to achieve the greatest speedup for the program?
Improving class A by a factor of 2 would save 4*20 = 80 million cycles
Improving class B by a factor of 2 would save 6*18 = 108 million cycles
Improving class C by a factor of 2 would save 5*12 = 60 million cycles
Improving class D by a factor of 2 would save 3*10 = 30 million cycles
Improving class E by a factor of 2 would save 2*6 = 12 million cycles

Hence class B should be improved.

d) (10) If only the class A instructions are improved in the original program, what improvement factor for class A is required to yield a speedup of 1.2608695 for the program?
Speedup = T_{before} / T_{after} = $(cycles_{before})$*cycle_time / $(cycles_{after})$*cycle_time
= $cycles_{before} / cycles_{after}$ = 580 / ((8/K)*20 + 12*18 + 10*12 + 6*10 + 4*6)
= 580 / ((160/K) + 420) = 1.2608695
So (160/K) + 420 = 580/1.2608695 = 460
160/K = 460 – 420 = 40
The required improvement factor K = 160/40 = 4
So class A must be improved by a factor of 4.

8. The sub-module on Computer performance metrics defines the geometric mean and the harmonic mean as alternatives to the arithmetic mean as techniques for comparing CPU performance. Some processors, especially those used in laptops or mobile devices, are able to throttle their speed as a means of conserving power and extending battery charge. Suppose that such a processor runs in the slower low power mode executes 2,890,000 instructions for a program in 14.45 milli-seconds. Next the processor switches to the higher speed mode and executes the same program again in only 7.225 milli-seconds. Answer the following questions and show how you obtained your answer for each:

a) (4) What is the MIPS rating for the first execution of the program?
MIPS for first execution = $(2890000 / 10^6) / (14.45*10^{-3}) = 200$

b) (4) What is the MIPS rating for the second execution of the program?
MIPS for second execution = $(2890000 / 10^6) / (7.225 *10^{-3}) = 400$

c) (4) What is the arithmetic mean of the MIPS ratings for the two executions of the program?
Arithmetic mean = (200 + 400)/2 = 300

d) (4) What is the geometric mean of the two MIPS ratings?
Geometric mean = square root(200 * 400) = 282.843

e) (4) What is the harmonic mean of the two MIPS ratings?
Harmonic mean = 2 / ((1/200) + (1/400))
= 2 * 200 * 400 / 200+400 = 160000/600 = 266.67

f) (4) What is the MIPS rating for the two executions of the program if it is computed based on the total combined number of instructions in the two executions of the program and on the total combined times consumed for the two executions of the program?
MIPS = $((2890000 + 2890000) / 10^6) / (14.45*10^{-3} + 7.225*10^{-3}) = 266.67$

Computer Science 605.611

Problem Set 2 Answers

1. (5) Every MIPS assembly language instruction that corresponds to a builtin true-op instruction can be translated into a single 32-bit machine instruction. Use 8 hex digits to show the 32-bit machine instruction that corresponds to the following MIPS assembly language instruction: sll $0,$0,0
Looking up this instruction in appendix A reveals that it is an R-type shift left logical instruction with opcode=0. The rs and rt registers are both 0 and the shift amount is 0. The unused fields within the machine instruction are 0. Hence the 32-bit machine instruction in hex is 0x00000000.

2. (5) Register $8 contains the 32-bit pattern corresponding to 0x8D6A0000. To what value (in decimal format ±d.dd * 10^E) does the pattern correspond if interpreted as an IEEE-754 floating point number? Express your answer to two decimal places in the format ±d.dd*10^E.
The corresponding binary pattern is 1 00011010 11010100000000000000000
So the sign is negative, the characteristic is decimal 26 which corresponds to an exponent of 26-127 = -101. Therefore the pattern represents – 1.110101 * 2^-101 = -1.D4 * 2^-101 = -1D4 *2^-8 * 2^-101 = - 468 * 2^-109 = - 7.21 * 10^-31.

3. (5) Recall that the excess-B representation of a signed integer N is given by N+B, where B is the bias. Use 8 hex digits to show the excess-2147483648 representation of the negative decimal integer -1305464520 . Where 2147483648 is the bias.
-1305464520 + 2147483648 = 842019128 = 0x32303138 .

4. (15) If register $8 (also referred to as $t0) contains the 32-bit pattern 0x34789602, indicate (yes or no) whether each of the following interpretations is a valid interpretation of this pattern.
a) a one's complement integer (if yes, write the integer in decimal) Yes. Since the MSB is 0 this represents a positive value +880317954. If the MSB had been 1, we would complement the pattern to determine what it represents.

b) a two's complement integer (if yes, write the integer in decimal) Yes. Since the MSB is 0 this represents a positive value +880317954. If the MSB had been 1, we would complement the pattern to determine what it represents. Note that when the MSB is 0 the value has the same representation in both two's and one's complement.

c) a 32-bit IEEE 754 floating point number (if yes, write the number in decimal)
Yes. The MSB is 0 so the sign is positive. The next 8 bits (01101000) give the excess-127 exponent = 104 – 127 = -23. The next 23 bits (11110001001011000000010) give the fraction. So the value represented is + 1.F12C04 * 2^-23
= 1F12C04 * 2^-24 * 2^-23 = 1F12C04 * 2^-47 = 32582660 * 2^-47 = 2.315 * 10^-7

d) a MIPS machine instruction (if yes, show the assembly language instruction)
The 6-bit opcode = 001101 = 0xD so the instruction operator mnemonic is ori (from appendix A). This is an I-type instruction. The rs field contains 00011. The rt field contain 11000. The remaining 16 bits give the immediate operand. So the corresponding assembly language instruction is ori $24,$3,0x9602. So the answer is yes.

e) a sign & magnitude integer (if yes, write the integer in decimal)
Yes. The sign bit is 0 so this represents a positive integer = +880317954.
Note that positive integers have the same 32-bit representation in one's complement, two's complement and sign & magnitude systems. The representations only differ for negative integers.

5. (5) a) Register $4 contains the 32-bit pattern 0x3E000002, which in binary is 00111110000000000000000000000010. What does the CP1 (coprocessor 1) register $f4 contain after the instruction mtc1 $4,$f4 is executed? Express your answer in hex. $f4 = 0x3E000002
This CPU instruction simply copies the bit pattern without change from CPU register $4 to CP1 register $f4.

(5) b) CP1 register $f6 contains the IEEE 754 representation of the negative decimal value -2.5. Use 8 hex digits to show the result in register $f8 after executing the instruction cvt.w.s $f8,$f6 .
This instruction uses truncation on $f6 to produce a whole number and places the two's complement representation of the integer into $f8. So the pattern produced in $f8 represents the negative value -2. So $f8 contains 0xFFFFFFFE.

6. (5) a) The jump instruction j loop transfers control to the instruction to which the label "loop" is attached. If the machine code for this jump instruction is located at memory address 0x40CE88C0. What is the highest address to which the label "loop" can correspond if this jump instruction is to work properly? Express you answer in hex.
The jump instruction shifts the rightmost 26 bits in the machine instruction left 2 bits (i.e., multiplies it by 4) and prepends the upper 4 bits of the PC to the resulting 28 bits to generate the 32-bit jump address. Hence the highest address to which the label loop can correspond is 0x4FFFFFFC.

(5) b) The conditional branch instruction beq $9,$8,exit transfers control to the instruction to which the label "exit" is attached if registers $9 and $8 contain equal values. If the machine code for this beq instruction is located at memory address 0x40CE88C0. What is the lowest address to which the label "exit" can correspond if the instruction is to work properly? Express you answer in hex.

Conditional branch instructions such as beq use the rightmost 16 bits in the machine instruction as a signed number that indicates the number of instructions to branch forward (if positive) to higher addresses or backwards (if negative) to lower addresses. The most negative 16-bit number is -32768, so the instruction can branch backwards up to 32768 instructions. At runtime, this 16 bit displacement is shifted left 2 bits (i.e., multiplied by 4) and sign extended to 32 bits. The 32-bit signed integer is then added to the already incremented PC to generate the branch destination address (also called the branch target address). The most negative displacement, when added to the PC, generates the lowest target address. Hence the lowest target address is 0x40CE88C4 + (- 32768*4) = 0x40CE88C4 – 0x20000 = 0x40CC88C4.

7. Appendix A contains the description of each of the instructions mentioned below. Use those descriptions to answer the following instructions:

(5) a) What 32-bit pattern (expressed using 8 hex digits) is placed into register $5 by the instruction lui $5,0x9AE3 ? Register $5 initially contains the two's complement representation of -2.

This instruction places 0x9AE30000 into $5. The low 16 bits are zeroed out.

(5) b) What 32-bit pattern (expressed using 8 hex digits) is placed into register $5 by the instruction addi $5,$5,0x9AE3 ? Register $5 initially contains the two's complement representation of -2.

This instruction adds the sign extended immediate operand to $5. Hence $5 will contain 0xFFFFFFFE + 0xFFFF9AE3 = 0xFFFF9AE1.

(-2 + -25885) = -25887

(5) b) What 32-bit pattern (expressed using 8 hex digits) is placed into register $5 by the instruction addiu $5,$5,0x9AE3 ? Register $5 initially contains the two's complement representation of -2.

The add immediate unsigned still adds the sign extended immediate to register $5. Hence register $5 will contain 0xFFFF9AE1 in this case as well.

(5) c) What 32-bit pattern (expressed using 8 hex digits) is placed into register $5 by the instruction srl $5,$5,3 ? Register $5 initially contains the two's complement representation of -2.

This is a logical right shift so $5 will contain 0x1FFFFFFF.

(5) c) What 32-bit pattern (expressed using 8 hex digits) is placed into register $5 by the instruction sra $5,$5,3 ? Register $5 initially contains the two's complement representation of -2.

This is an arithmetic right shift, so the sign is preserved by bringing in copies of the original sign bit on the left. Register $5 will contain 0xFFFFFFFF.

8. (9) The variable name X corresponds to a 32-bit memory word that contains some integer in two's complement form. The address of the memory word is 0x100400CC. Write down a series of instructions (containing only MIPS true-ops) that adds the contents of the variable X to the contents of register $5 and places the sum back into register $5. Recall that each built-in true-op instruction (unlike pseudo-instructions) corresponds to a single machine instruction.

```
lui     $1,0x1004
lw      $1,0xCC($1)
add     $5,$5,$1
```

9. (6) Show the true-op assembly language instructions to which the MIPS pseudo-instruction `la $10,Y` corresponds. This pseudo-instruction places the address of Y into register $10. Assume that the address of Y is 0x87659324.

```
lui     $10,0x8765
ori     $10,$10,0x9324
```

b) (5) Why is the instruction `lw $4,X` not a valid MIPS true-op instruction?
Instructions that reference memory operands must use a base register and displacement to generate the operand address. There is no direct addressing for memory operands since the 32-bit address will not fit into the rightmost 16 bits of the instruction.

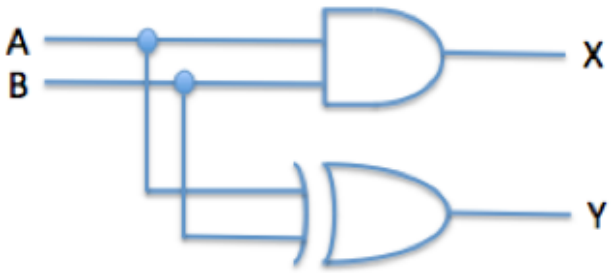
10. (5) What 32-bit pattern (expressed using 8 hex digits) is placed into register $5 by the instruction `xori $5,$5,0x9AE3`? Register $5 initially contains the two's complement representation of -2.
The logical instructions (unlike the arithmetic) integer instructions zero extend the 16-bit immediate operand to a 32-bit pattern. Hence the result produced in register $5 is 0xFFFFFFFE XOR 0x00009AE3 = 0xFFFF651D .

Computer Science 605.611

Problem Set 3 Answers

[For this problem set, use "*" to denote logical AND, "+" to denote logical OR, "^" to denote exclusive-OR and the apostrophe " ' " to denote NOT (e.g., N' means NOT N).]

1. a) (5) Complete the truth table below by filling in the columns for the circuit outputs Y and X as a function of the inputs A and B.



A	B	Y	X
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

b) (5) To what arithmetic functions do the outputs Y and X correspond?

Y corresponds to the single-bit arithmetic sum of A + B and X corresponds to the carry generated when the sum of A+B is computed.

2. (10) Consider the following truth table that defines the output C as a function of the two inputs A & B:

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Use only Boolean identities (not truth tables) to show that the logical sum of the non-zero minterms for this function is equivalent to the logical product of the zero-maxterms.

The logical sum of the non-zero minterms is:

(A' * B) + (A * B')

The logical product of the zero maxterms is:

(A + B) * (A' + B')

Applying the distributive law to this yields

A*(A' + B') + B*(A' +B') = A*A' + A*B' + B*A' + B*B' = 0 + A*B' + B*A' + 0

= A'*B + A*B'

3. Recall that the modulo function L MOD N is defined as the remainder produced when L is divided by N.

a) (5) Write down a series of MIPS true-op instructions that use the integer divide instruction to produce in register $8, the result of the function $8 MOD $4.

```
div  $8,$4      ;divide $8 by $4, lo reg contains the quotient and hi contains the remainder
mfhi  $8        ; copy remainder from hi into $8
```

b) (5) Write down a single (i.e., one) MIPS logical instruction that produces in register $9 the result of the function $9 MOD 256.

```
andi    $9,$9,255      ; clear all but the low 8 bits in $9
```

4. (5) Consider the circuit:

Write down an equivalent logic function involving only the operators * for AND, + for OR and ‘ for NOT. X = __A’ * B’ +A * B___

5. An encoder generates an output that identifies which one of its inputs is active. The output is in effect the index of the single active input.

a) (5) How many outputs are required for an encoder that has 8 inputs?
Since there are 8 inputs, the index of the active input can range from 0 to 7 (000 to 111 in binary). So three output bits are required.

b) (5) If a decoder like that described in module 3 has 4 outputs, how many inputs should it have? The decoder interprets its inputs as an N-bit number and asserts the one output whose index is given by the N-bit number. Since there are 4 outputs, the input index only ranges from 0 to 3. So two binary inputs are required.

6. (5) Suppose the register $7 contains a single precision IEEE 754 floating point number. Write down a series of MIPS integer instructions that changes the floating point number in $7 into its arithmetic negative.

Inverting the sign bits negates the floating point number. Hence the following pair of instructions will work:

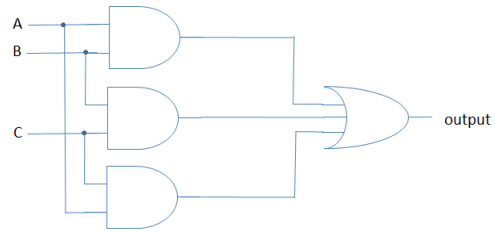
```
lui     $1,0x8000

xori    $7,$7,$1
```

7. (5) How can the same full adders used to generate the sum of two integers N+M be used instead to generate their difference N-M ?

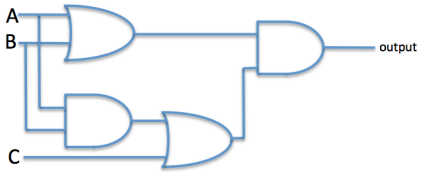
This can be done by adding the two's complement negative of M to N. So the same circuits used for addition can be used for subtraction.

8. a) (5) Write down a logic expression for the output generated by the following circuit:



output = AB + BC + AC

b) (5)Write down a logic expression for the output generated by the circuit below:

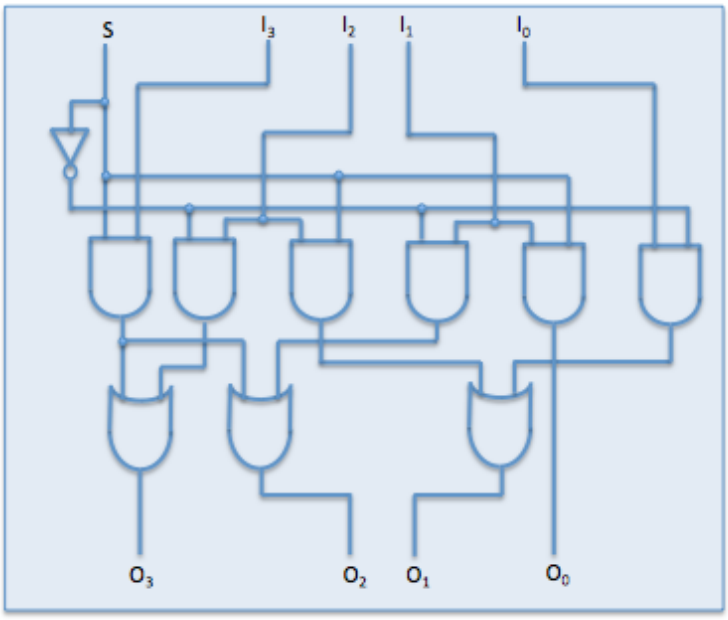


output = (A+B)(AB +C)

c) (5) Use Boolean identities (not truth tables) to show that the expression you obtained for part b) can be transformed into the expression you obtained for part a). That is, the expression for part b) can be rewritten as the expression for part a).

output = (A+B)(AB +C) = A(AB+C) + B(AB+C)
= AAB + AC + BAB + BC = AAB + AC + ABB + BC = AB + AB + AC + BC
= AB + BC + AC

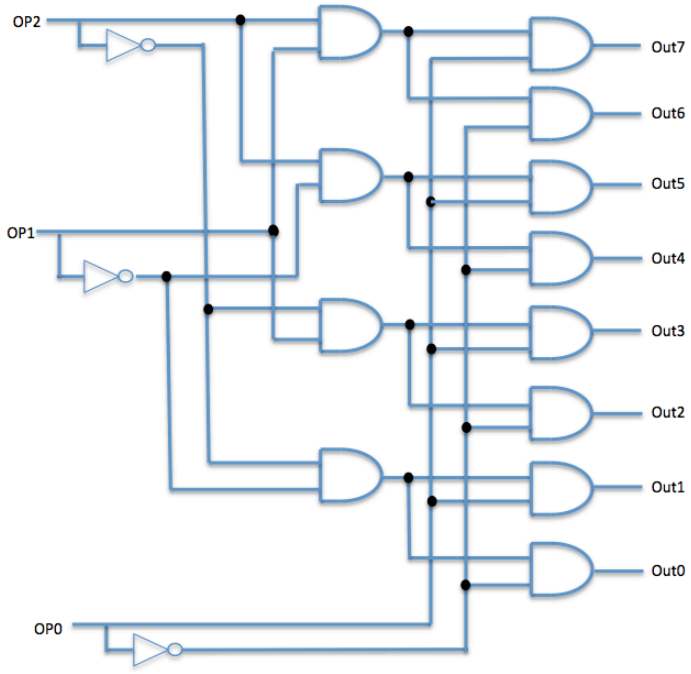
9. Consider the following circuit:



a) (5) If S=0 show the outputs if initially I_3, I_2, I_1 I_0 = 0101.
O_3 = ___1___
O_2 = ___0___
O_1 = ___1___
O_0 = ___ 0___

b) (5) If S=1 show the outputs if initially I_3, I_2, I_1 I_0 = 1110.
O_3 = ___1___
O_2 = ___1___
O_1 = ___1___
O_0 = __ 1____

10. (10) Consider a CPU that uses 3-bit opcodes. Draw a logic circuit containing only discrete NOT gates together with AND each of which uses only 2 inputs; the circuit should implement a decoder for the 3-bit opcode. Your circuit should contain the minimum number of NOT gates and the minimum number of 2-input AND gates.
A minimum of 3 inverters are needed (one for each of the 3 inputs).
Four AND gates are required to generate (Op2*Op1), (Op2*Op1'), (Op2'*Op1) and (Op2'*Op1'). Four more AND gates can be used to AND each of these terms with OP0; another four AND gates can be used to AND each of these with OP0'. So a total of 4+4+4 = 12 AND gates are required. The corresponding circuit is shown below:



11. (5) The following boolean expression defines Z as a function of four inputs A, B, C and D:
Z = A*D*(B+C) + B*C*(A + D)

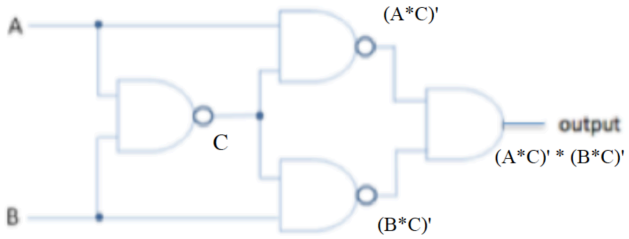
where "*" denotes a logical AND operator and "+" denotes a logical OR operator. Which of the following types or circuits best matches this function? Explain why.

I. Comparator (Z=1 if all the inputs are equal)
II. Majority vote (Z=1 if more than half of the inputs are set)

This is a majority vote function since the output is 1 only if at least 3 of the 4 inputs are 1. That is, if the majority of the inputs are 1. Otherwise the output is 0.

A comparator outputs 1 if all of the inputs are the same and outputs 0 if the inputs differ. This circuit outputs 1 even if, for example, B=0 and the other inputs are all 1. Hence it is not a comparator

12. (5) Consider the logic circuit shown below:



Which one of the following individual logic gates is equivalent to this circuit (i.e., generates the same output for the same inputs)?

a) AND
b) OR
c) NOR
d) NAND
e) XNOR
f) XOR

Let C=(A*B)'
output = (A*C)' * (B*C)' = (A' + C') * (B' + C') = A' * (B' + C') + C' * (B' + C')
= A' * B' + A'*C' + C' * B' + C' * C' = A' * B' + A' * C' + C' * B' + C'
= A' * B' + A' * C' + C' * (B' + 1) = A' * B' + A' * C' + C' = A' * B' + C' * (A'+1)
= A' * B' + C' = A' * B' + A * B = A XNOR B by definition. So the answer is e)

Computer Science 605.611

Problem Set 4 Answers
(Please do not use outside resources such as simulators, disassemblers or web sites to complete this assignment.)

1. (4) Consider the following statement: "Every MIPS assembly language statement is translated by the assembler into a single 32-bit machine code instruction." Explain why the statement is true or is false.
The statement is false because an assembly language statement may be a pseudo-instruction that corresponds to multiple true-ops or may be an assembler directive that causes the generation of an amount of data other than a single 32-bit word.

2. a) (7) Prior to executing the MIPS machine instruction 0x10A1FFFE (which resides at address 0xB0000000), CPU registers $1 through $31 all contain the pattern 0xFFFFFFFA. What is the hex pattern in the PC (program counter) and any modified CPU general purpose registers ($0 through $31) immediately after this instruction is executed?
The 6-bit opcode=000100, so this is a beq instruction that branches if the contents of the rs register ($5) and the rt register ($1) are the same. Since the two registers contain the same value, the branch is taken and the PC is set to the branch target address computed as the incremented PC contents plus 4 times the sign-extended low 16 bits of the machine instruction.
The 16-bit offset 0xFFFE represents -2 (indicating a backwards branch of 2 instructions). 4 * (-2) = -8
So the branch target address = 0xB0000004 – 8
= 0xB0000004 + 0xFFFFFFF8 = 0xAFFFFFFC.

b) (7) Prior to executing the MIPS machine instruction 0x04A1FFFE (which resides at address 0xB0000000), registers $1 through $31 all contain the pattern 0xFFFFFFFA. What is the hex pattern in the PC (program counter) and any modified CPU general purpose registers ($0 through $31) immediately after this instruction is executed?
The 6-bit opcode=000001, so this is a conditional branch instruction that compares the rs register with the constant 0. The rt field (=00001) indicates that this is the instruction bgez which branches if the contents of the rs register is greater than or equal to 0. The rs field contains 00101 indicating register $5. Since register $5 contains 0xFFFFFFFA (= - 6), the branch is not taken and the PC will contain the address of the next instruction (0xB0000004).

3. (4) For our MIPS 9-instruction core subset (add, sub, and, or, slt, lw, sw, beq & jump), which bits within the machine instruction does the control unit examine to decide whether to set the RegDst control bit to 0 or to 1?
For the MIPS core instruction subset, the control unit examines bits 31 through 26 (the opcode). If these opcode bits = 000000 (R-type instruction) then RegDst is set to 1. Any other MIPS core instruction opcode will cause RegDst to be set to 0.

4. (4) List all assembly instructions within our 9-instruction MIPS core subset for which the control unit would set the MemtoReg control bit to 1 when it processes the corresponding machine instruction.
MemtoReg is set to 1 for instructions whose result comes from the data memory. The only instruction within our 9-instruction subset for which this is the case is the lw instruction.

5. (5) Suppose the instruction jr $t0 resides at memory address 0x04000000. What is the highest valid instruction address (expressed in hex) to which this instruction can transfer control?
This instruction uses a register to contain a full 32-bit destination address. Instruction addresses must be a multiple of 4. The highest 32-bit multiple of 4 is 0xFFFFFFFC.

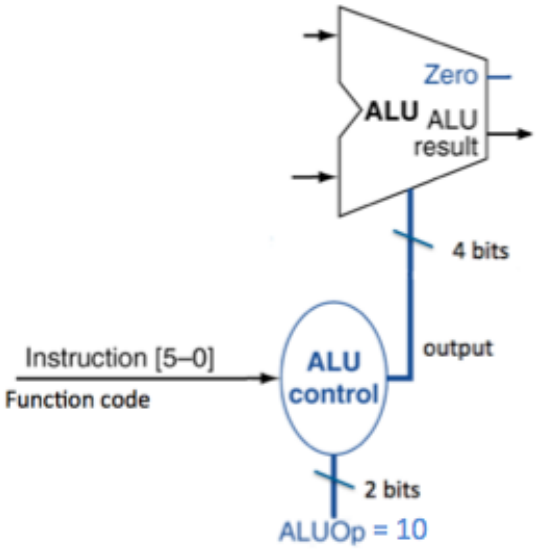
6. (5) List all assembly instructions within our MIPS 9-instruction core subset whose machine instruction format is the I-type format.
The beq, lw & sw instructions all use the I-format.

7. (4) The instructions: addi, ori, and lw all use the register identified by the rt field as the result register. Why can't these instructions use the register identified by the rd field as the result register?
These are all I-type instructions that use bits 15 – 11 within the machine instruction as the high bits of the 16-bit immediate field instead of specifying the rd field. These machine instructions contain only an opcode, rt, rs and immediate field.

8. (6) The diagram below shows how the ALU control unit directs the ALU during the execution of the instructions within our MIPS core subset. Assuming that the ALUOp bits = 10, complete the table below to show the 4-bit output from the ALU control unit for each of the listed 6-bit function code inputs described in module 4:

Function code	4-bit ALU control output
101010 (slt)	0111
100100 (AND)	0000
100000 (add)	0010
100101 (OR)	0001
100010 (subtract)	0110

These bit patterns were specified in the ALU control submodule within module 4.



9. a) (4) Why does the single-cycle datapath require separate instruction and data memories?
All instructions must be fetched from memory and instructions such as lw and sw also reference the data memory. A memory can only perform one operation (read or write) at a time. Therefore separate instruction and data memories are required to allow the instruction fetch and memory operand access to occur within the same clock cycle.

b) (4) Instead of using a single control bit Memctl to indicate whether to perform a memory read (Memctl =0) or a memory write (Memctl=1) with the single-cycle datapath, why are separate control bits (MemRead and MemWrite) used?
If only a single bit were used, then every instruction would be forced to either read from memory or write to memory. MemRead and MemWrite are both required so that when both bits are 0, neither a memory read nor a memory write occurs (as for the R-type instructions).

9. (5) Assume that $6 contains a two's complement integer that is greater than the two's complement integer in $5. After the instruction sub $4,$5,$6 is executed, what is the value of the MSB (bit31) in register $4 if an overflow occurs?
Subtracting a larger value from a smaller value should produce a negative result. A negative result will have a 1 in the MSB. However if an overflow occurs, the result will incorrectly appear to be non-negative. So the MSB = 0 if an overflow occurs.

10. (5) List all instructions within our MIPS core instruction subset (add, sub, and, or, slt, lw, sw, beq & jump) that do not require the output of the sign extension unit.
The instructions that use the output of the sign-extension unit are beq, sw and lw. The instructions that do not use the output of the sign-extension unit are add, sub, and, or, slt & j.

11. (10) Assume that the zero flag and registers $0 through $31 all contain 0 prior to executing the machine instruction 0x0C3A7E24 which resides at memory address 0x400B00C8. What value (expressed in hex) will be in any CPU registers that are modified and what value will be in the program counter (PC) register when the instruction completes execution?

The opcode corresponds to the leftmost 6 bits (000011 binary = 3), so this is a jal (jump and link) instruction (J-type). The jump address is produced by taking the upper 4 bits from the PC (0x4 in this case) and appending the rightmost 26 bits from the instruction shifted left 2 positions. Shifting the rightmost 26 bits left 2 positions yields the 28-bit pattern: 0x0E9F890 = (0x3A7E24 << 2). Prepending 0x4 (the leftmost 4 bits of the PC) to this pattern produces the 32-bit jump address 0x40E9F890, which is placed into the PC.

The address of the next instruction is placed into register $31 (the link register) as the return address. This return address is 0x400B00CC (or 0x400B00D0 taking into account the branch delay slot (which will be explained when pipelining is covered).

12. Show the 8-digit hex representation of the 32-bit machine instruction for each of the following true-op assembly language instructions:

(3) add $4,$5,$6 machine code = 000000 00101 00110 00100 00000 100000 = 0x00A62020

(3) or $4,$5,$6 machine code = 000000 00101 00110 00100 00000 100101 = 0x00A62025

13. Consider the MIPS assembly language instruction ori $6,$4,0xC8 .

a) (3) What is the operator mnemonic for this instruction?

The operator mnemonic is the name of the operation "ori'.

b) (3) What is the opcode for this instruction?

The leftmost 6 bits in the machine instruction contain the opcode = 001101 = 0x0D.

14. Suppose that registers $12 and $11 both contain the two's complement representation of the negative decimal integer -2147483648.

a) (3) What decimal value is represented by the result produced by the instruction: addiu $5,$12,-2 ? Decimal value = _____

This instruction adds the sign extended immediate operand plus $12 to produce the sum 0x80000000 + 0xFFFFFFFE = 0x7FFFFFFE in register $5. This pattern represents decimal +2147483646. Note that signed as well as unsigned arithmetic instructions sign extend the immediate operand.

b) (3) What decimal value is represented by the result produced by the instruction: andi $5,$11,-2 ? Decimal value = _____

This is a logical instruction. Logical instructions, unlike arithmetic instructions, zero-extend their immediate operand to 32 bits. So the pattern produced in register $5 is 0x80000000 AND 0x0000FFFE = 0x00000000. This pattern represents decimal 0.

c) (2) Does either or both of these instructions cause an overflow exception? If so, identify the instruction(s). The addi instruction causes an overflow since its result is interpreted as a signed integer. Neither the addiu instruction nor the andi instruction causes an overflow exception.

15. (6) Suppose that a hardware error, hack attack, or bit flip due to cosmic radiation inverts bit 20 within the machine instruction for bgez $4,exit. Bit 20 is the only bit affected. Would the modified 32-bit machine instruction still correspond to a valid instruction? If so, what is the assembly language instruction, when translated to machine code, corresponds to the modified 32-bit pattern?

Bit 20 within the machine instruction for bgez $4,exit is part of the rt field (00001). Inverting the bit changes the field to 10001. This makes the resulting 32-bit pattern appear to be the machine code for the assembly language instruction:
bgezal $4,exit.

Computer Science 605.611

Problem Set 5 Answers

1. Suppose that our MIPS core instruction subset (add, sub, slt, and, or, lw, sw, beq & j) is expanded to include another instruction, the or immediate (ori) instruction. Answer each of the following questions about possible modifications to the multi-cycle datapath and FSM that may be required to support this additional instruction.

a) (3) What is the minimum number of new states required for the FSM (finite state machine) description of the ori instruction execution?
Two new states would be required: state 10 in which the OR of the zero-extended immediate with the rs register is performed; and state 11 in which the ALU output is written into the rt register (not the rd register as in state 7). The bits that normally identify the rd register (bits 11 through 15) are part of the immediate operand field for I-type instructions.

b) (3) Identify any new control bits that the control unit has to output for the ori instruction.
The lower B input to the ALU would now have 5 possible candidates: the constant 4, the rt register, the sign-extended immediate, the sign-extended immediate shifted left 2 bits & the zero-extended immediate. So ALUSrcB would have to be expanded from 2 bits to 3 bits (by including one new bit) and a larger MUX is needed to select from among the 5 inputs. If instead the sign-extension unit is modified to also produce a zero-extended immediate, a new control bit is needed to tell the modified extension unit which type of immediate to generate.

c) (3) For each new state required for ori, list all control bit settings used in the new state and indicate what determines when to transition into the new state.
In the new state 10, the 3-bit ALUSrcB should be set to 100 to select the zero-extended immediate as the lower ALU B input, ALUSrcA=1, and ALUOp should be set to the unused pattern 11 to denote an OR operation. State 10 is entered following state 1 based on the ori opcode. State 11 follows state 10. In state 11, RegDst=0, RegWrite=1 & MemtoReg=0 to write the ALU output to the rt register. State 7 can't be used since it sets RegDst=1 which selects rd as the result register.

d) (3) Identify any new hardware devices that are required in the datapath to support the execution of the ori instruction.
A zero-extension unit is required. Alternatively, the existing extension unit could be modified to generate either the signed-extended or the zero-extended immediate. This would require a new control bit to tell the extension unit to sign or zero extend.

2. The micro-programmed implementation of our MIPS multi-cycle datapath employs a micro-PC.
a) (3) How many bits does the micro-PC contain?
The micro-PC requires 4 bits to indicate the state number (0 through 9).
Each state corresponds to a separate micro-instruction.

b) (3) What do the bits within the micro-PC indicate? They indicate which micro-instruction to use next (i.e., the index of the micro-instruction).

c) (3) When the micro-PC is incremented, by how much is it incremented?
Unlike the PC, the micro-PC is incremented by 1.

d) (3) When the micro-PC is updated, what bits within the micro-instruction indicate how to update the micro-PC?
The 2 LSBs address control (AdrCtl) indicate how to update the micro-PC (set it to 0, increment it, or use ROM1 or ROM2 LUT (lookup table) to obtain the next value for the micro-PC).

3. Both of our MIPS single-cycle and multi-cycle datapaths use control bits to indicate the operations and direct the actions that take place when executing machine instructions.
a) (3) How many control bits are required for our MIPS single-cycle datapath?
There are 10 control bits for the single-cycle datapath to support our MIPS core instruction subset.

b) (3) How many control bits are required for our MIPS multi-cycle datapath?
There are 16 control bits required for the multi-cycle datapath.

4. If the control unit for the single-cycle datapath were implemented using a PLA (programmable logic array), what is the minimum number of inputs and the minimum number of outputs required for the PLA? Name each of the inputs and the outputs.
a) (5) minimum inputs are: _4_the 6 opcode bits (op5 through op0) identify the type of instruction. However for the core subset Op0 through Op3 are sufficient to distinguish one instruction from the other in the subset.

b) (5) minimum outputs are: ___there are 10 datapath control bits (ALUOp1, ALUOp0, RegDst. RegWrite, MemtoReg, ALUSrc, MemRd, MemWrt, Branch, Jump) required of our MIPS core instruction subset._____

5. a) (3) Why do CISC processors tend to use micro-programming?
CISC systems tend to use micro-code because it is easier to use micro-programming to implement the more complex instructions.

b) (3) Why do RISC processors not use micro-programming?

RISC systems do not use micro-code because hardwired logic is faster. Multiple micro-instructions must be retrieved and interpreted for each machine instruction while hardwired RISC systems execute the machine instructions directly.

c) (3) What is the main reason that ROM rather than RAM is used to store micro-code (i.e., micro-programs)?
Using ROM avoids the possibility of corrupting the micro-code through accidental or intentional attempts to modify the micro-code. Also the micro-code is retained in ROM even when the power is off. RAM is volatile and loses its contents when the power is off.

6. Answer each of the following questions about the micro-programmed implementation of our MIPS multi-cycle datapath described in module 5:

a) (3) How many bits are contained within each machine instruction? All machine instructions contain 32 bits.

b) (3) How many bits are contained within each micro-instruction?
Each micro-instruction contains 18 bits (the 16 control bits and the 2 AdrCtl bits).

c) (3) What is the total number of micro-instructions contained in the control memory for the micro-programmed system for our MIPS core instruction subset?
The control memory contains 10 micro-instructions (one for each of the 10 states).

d) (5) Use hex to show each micro-instruction in the sequence of micro-instructions (i.e., the micro-program) needed for the add machine instruction.
These correspond to the FSM states 0, 1, 6 and 7.

State number	Control word bits 17–2	Control word bits 1–0
0	1001010000001000	11
1	0000000000011000	01
2	0000000000010100	10
3	0011000000000000	11
4	0000001000000010	00
5	0010100000000000	00
6	0000000001000100	11
7	0000000000000011	00
8	0100000010100100	00
9	1000000100000000	00

0x25023
0x00061
0x00113
0x0000C

7. Answer the following questions about the finite state machine (FSM) model of our muti-cycle datapath control shown in module 5:

a) (3) How many separate states are there in the FSM?
There are 10 states (0 through 9).

b) (3) How is the opcode within each machine instruction used by the FSM?
The opcode is used in state 1 and in state 2 to determine which state to enter next.

c) (3) What is the maximum number of states used in the FSM for any of the instructions within our MIPS core instruction subset?
The maximum number is 5 used for the lw instruction.

d) (3) Is the FSM an example of a Mealy machine or an example of a Moore machine (as defined in module 5 CU_implementation)?
The outputs are the control bits, which depend only on the current state, so the FSM is a Moore machine.

8. The ALU shown in the multi-cycle datapath diagram in module 5 has two data input ports: the upper A input port and the lower B input port.

a) (5) Name each of the possible A inputs to the ALU?
The 2 possible A inputs are the rs register and the PC register.

b) (5) Name each of the possible B inputs to the ALU?
The 4 possible B inputs are the constant 4, the rt register, the sign-extended immediate and the sign-extended immediate shifted left 2 bits.

9. (5) Our MIPS ALU outputs a single-bit flag called "zero" that is set to 1 to indicate when the ALU result is 0. Suppose that the ALU also output three other flags which, when set, indicate one of the following conditions:

- N to indicate a negative result
- V to indicate an arithmetic overflow
- C to indicate a carry

After the instruction sub $t0,$t1,$t2 subtracts $t2 from $t1 and places the result into $t0, some combination of the flags (Z, N, V, C) indicates that $t1 contained a value less than that in $t2? Write down the logic expression that shows the required combination of Z, N, V and C in simplest form.
Logic expression = N * V' + N' * V = N XOR V.
If there is no overflow, the result must be negative, or positive with an overflow. We know, of course, that if $t1 < $t2 then $t1 - $t2 < 0. However a signed overflow is possible which causes the sign of the value in the result register $t0 to be incorrect.

10. You know that every instruction takes one clock cycle on the single-cycle datapath. How many clock cycles are required to fetch and execute each of the following instructions on our MIPS multi-cycle datapath?

a) (2) sw 4 cycles

b) (2) add 4 cycles

c) (2) slt 4 cycles

d) (2) lw 5 cycles

e) (2) beq 3 cycles

This can be determined from the FSM for the multi-cycle datapath.

Computer Science 605.611

Problem Set 6 Answers

1. The table below lists the minimum time required to complete all activity performed in each stage within our MIPS 5-stage pipeline:

Pipeline Stage	Time required
Fetch	12.5 ns
Decode	8.5 ns
Execute	11.5 ns
Memory	12.5 ns
Write-back	8ns

a) (3) Based on this information, what is the maximum clock rate that can be used for the pipeline?

The clock period has to be as long as the most time consuming stage.
So the maximum clock rate = 1/12.5ns = 80 MHz.

b) (3) In which pipeline stage does the instruction `add $8, $4, $2` read $4 ?

All instructions read their input registers in stage 2 (the decode stage) during the second half of the clock cycle.

c) (3) In which pipeline stage does the instruction `add $8, $4, $2` write $8 ?

Result registers are written in the write-back stage (stage 5) during the first half of the clock cycle.

d) (3) In which pipeline stage does the instruction `lw $8, 4($2)` compute the memory address of the word that it reads from memory?

In the execute stage (stage 3) the ALU is used to add the sign-extended displacement to the base register ($2 in this case) to produce the address of the word that is read from memory by the lw instruction in stage 4 (the memory stage).

e) (3) In which pipeline stage does the instruction `beq $9,$6,exit` compute the branch target address?

The branch target address is computed in stage 3, the execute stage using a separate adder to compute the sum of the PC contents plus the sign-extended immediate field shifted left two bits.

2. (6) What is the difference between a data dependency and a data hazard with respect to the pipeline?

A data dependency exists when an instruction uses an input operand that is written by another instruction ahead of it in the pipeline.
A data dependency is a data hazard only if the dependent instruction attempts to read the required result before the instruction that produces the result writes it.

3. Assume that the 5-stage MIPS pipeline runs at a clock rate of 0.05 GHz.

a) (3) With no data hazards, how many nano-seconds does the instruction lw $8, 4($2) take to go through the pipeline?

If there are no hazards then no stalls are required. The clock rate of 0.05 GHz corresponds to a cycle time of 1/0.05GHz = 20 ns.
Since each of the 5 stages must take 20ns, at least 5*20ns = 100ns are required for the lw instruction to go through the pipeline.

b) (3) With no data hazards, how many nano-seconds does the instruction add $8, $4, $2 take to go through the pipeline?

If there are no hazards then no stalls are required. However, every instruction must go through all 5 pipeline stages, so this instruction takes 100 ns as well.

4. a) (3) What is the maximum number of R-type instructions that our original 5-stage MIPS pipeline completes per second if it runs at 4 GHz?
A 4 GHz clock rate corresponds to a cycle time of 1/4GHz = 0.25 nano-seconds. The pipeline can complete at most 1 instruction per cycle. One instruction every 0.25 nano-seconds corresponds to 4 billions instructions per second.

b) (3) What is the maximum number of R-type instructions that our non-pipelined multi-cycle datapath completes per second if it runs at 4 GHz?
On the multi-cycle datapath, each R-type instruction requires 4 cycles. One instruction every 4 cycles corresponds to one instruction every 4*0.25 nano-seconds = 1 nano-second, or 1 billion instructions per second.

5. a) (4) Once a pipeline bubble is created to cope with a data hazard, what is the minimum number of stages through which the bubble must travel?
Pipeline bubbles are stages in which nothing happens which appear to be empty. Setting all 9 control bits to zero results in no action being taken within the stage containing the bubble. Bubbles are created in the decode stage and travel through the final 3 stages of the pipeline.

b) (4) What is the minimum number of pipeline stages through which a nop must travel?
A nop is a 32-bit machine instruction (it corresponds to the instruction sll $0,$0,0). Nops are fetched from memory like any other machine instruction and must travel through all 5 pipeline stages.

c) (4) Is a nop created by the assembler or is it created by the control unit?
The assembler generates a nop when it translates the instruction with the mnemonic “nop” into a machine instruction that corresponds to sll $0,$0,0 .

d) (4) Is a pipeline bubble created by the assembler or is it created by the control unit?

The control unit creates bubbles in stage 2 by outputting 9 zero control bits. Four of the bits (ALUSrc, RegDst, ALUOp1, ALUOp0) affect the execute stage, three of the control bits (Branch, MemRead, MemWrite) affect the memory stage and two control bits (MemtoReg & Regwrite) affect the write-back stage.

6. (8) Recall that with our 5-stage pipeline, register reads occur in the second half of the clock cycle while register writes occur in the first half of the clock cycle. Consider the following instruction sequence:

```
xor     $2, $0, $3
slt     $5 ,$2, $4
add     $11, $5, $11
sllv    $6, $11, $12
lw      $8, 0x800($2)
sub     $2, $6, $8
```

Assume that the pipeline system employs a hazard detection unit but no other techniques, such as data forwarding or code rearrangement, to cope with data hazards. If the xor instruction is fetched in clock cycle 1, during which clock cycle does the sub instruction complete its write-back stage?

Without a forwarding unit, dependent instructions must be stalled in the decode stage until the instruction producing the required input reaches the write-back stage. Hence the instructions flow through the pipeline as follows:

Cycle	Fetch	Decode	Exec	Mem	Write-back
1	xor $2, $0, $3				
2	slt $5 ,$2, $4	xor $2, $0, $3			
3	add $11, $5, $11	slt $5 ,$2, $4	xor $2, $0, $3		
4	add $11, $5, $11	slt $5 ,$2, $4	bubble	xor $2, $0, $3	
5	add $11, $5, $11	slt $5 ,$2, $4	bubble	bubble	xor $2, $0, $3
6	sllv $6, $11, $12	add $11, $5, $11	slt $5 ,$2, $4	bubble	bubble
7	sllv $6, $11, $12	add $11, $5, $11	bubble	slt $5 ,$2, $4	bubble
8	sllv $6, $11, $12	add $11, $5, $11	bubble	bubble	slt $5 ,$2, $4
9	lw $8, 0x800($2)	sllv $6, $11, $12	add $11, $5, $11	bubble	bubble
10	lw $8, 0x800($2)	sllv $6, $11, $12	bubble	add $11, $5, $11	bubble
11	lw $8, 0x800($2)	sllv $6, $11, $12	bubble	bubble	add $11, $5, $11
12	sub $2, $6, $8	lw $8, 0x800($2)	sllv $6, $11, $12	bubble	bubble
13		sub $2, $6, $8	lw $8, 0x800($2)	sllv $6, $11, $12	bubble
14		sub $2, $6, $8	bubble	lw $8, 0x800($2)	sllv $6, $11, $12
15		sub $2, $6, $8	bubble	bubble	lw $8, 0x800($2)
16			sub $2, $6, $8	bubble	bubble
17				sub $2, $6, $8	bubble
18					sub $2, $6, $8

The sub instruction completes in cycle 18.

7. (5) A sequence of 21 MIPS instructions contains 10 R-type instructions, 5 lw instructions and 6 sw instructions. If there are no hazards of any type, what speedup would the 5-stage pipeline provide for this instruction sequence compared to the multi-cycle datapath with both datapaths running at the same clock rate? Express your answer to 2 decimal places.

On the multi-cycle datapath R-type instructions take 4 cycles, lw takes 5 and sw takes 4. Hence the total number of cycles consumed on the multi-cycle datapath is 10*4 + 5*5 + 6*4 = 89.

If there are no hazards, then no pipeline stalls or bubbles are needed. Hence the total number of cycles consumed on the pipeline is 5 + 20 = 25.

So the speedup provided by the pipeline is 89/25 = 3.56.

8. (5) A sequence of N instructions from our MIPS core instruction subset executing on the non-pipelined multi-cycle datapath achieves an average CPI of 3 cycles per instruction. Running this same sequence on our 5-stage pipeline with no hazards and at the same clock rate provides a speedup of 2.5. If the number of instructions in the sequence is doubled while keeping the average CPI the same, what speedup does the pipeline provide for this sequence of 2N instructions running at the same clock rate if there are no hazards? Round your answer to 2 decimal places.

For the sequence of N instructions the speedup = 2.5

= (cycle_time*N*3) / [cycle_time*(5+N-1)]

So N*3 = 2.5*(4+N)

N*3 – 2.5*N = 2.5*4

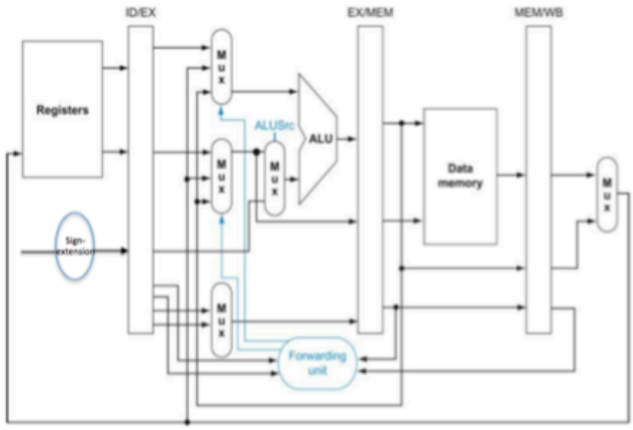
0.5*N = 10

N = 20

For the sequence of 2N instructions the speedup = (2*N*3) / (4 +2*N)

= 120 / 44 = 2.73

Base your answers to problems 9 through 11 on the version of the 5-stage pipelined datapath whose decode stage contains a data hazard unit and whose execute stage contains a forwarding unit is as shown below:



9. (8) Consider the following instruction sequence:

```
lui     $2, 0x3E4
lw      $8, 0x8($2)
add     $10, $2, $8
slt     $6, $10, $8
sw      $6,8($2)
sub     $2, $6, $8
```

Assume that the instructions are executed in the order shown. Complete the table below to show the instruction in each pipeline stage for each clock cycle until the entire instruction sequence completes. Do not transpose or otherwise rearrange the table. You may include as many additional rows as you need.

With a forwarding unit, any required inputs can be forwarded to the execute stage of dependent instructions eliminating one or more bubbles. However, a dependent instruction in a load delay slot (the slot immediately following a lw instruction) must be stalled in the decode stage until the lw completes its memory stage. Hence the instructions would flow through the pipeline as follows:

Cycle	Fetch	Decode	Exec	Mem	Write-back
1	lui $2,0x3E4				
2	lw $8,8($2)	lui $2,0x3E4			
3	add $10,$2,$8	lw $8,8($2)	lui $2,0x3E4		
4	slt $6,$10,$8	add $10,$2,$8	lw $8,8($2)	lui $2,0x3E4	
5	slt $6,$10,$8	add $10,$2,$8	bubble	lw $8,8($2)	lui $2,0x3E4
6	sw $6,8($2)	slt $6,$10,$8	add $10,$2,$8	bubble	lw $8,8($2)
7	sub $2,$6,$8	sw $6,8($2)	slt $6,$10,$8	add $10,$2,$8	bubble
8		sub $2,$6,$8	sw $6,8($2)	slt $6,$10,$8	add $10,$2,$8
9			sub $2,$6,$8	sw $6,8($2)	slt $6,$10,$8
10				sub $2,$6,$8	sw $6,8($2)
11					sub $2,$6,$8

10. The following short sequence of instructions runs on the pipelined datapath:

```
ori   $8,$0,0x1008
sll   $8$8,16
ori   $6,$0,4
sw    $6,4($8)
sw    $8,8($8)
lw    $6,0($8)
```

a) (3) When ori $6,$0,4 is in the execute stage, show what the following bit patterns should be for the datapath shown above:
ForwardA = __00__, ForwardB = __00__, ALUSrc = __1___
The ori $6,$0,4 instruction requires no forwarded operands, so ForwardA = 00, ForwardB=00 but ALUSrc=1 to select the constant 4 as its lower ALU input.

b) (3) When sw $6,4($8) is in the execute stage, show what the following bit patterns should be for the datapath shown above:
ForwardA = __01__, ForwardB = __10__, ALUSrc = __1___
The sw $6,4($8) requires the forwarded value for $8 (in the MEM/WB pipeline register produced by the sll instruction) as the upper ALU A input and requires the constant 4 as the lower ALU B input. Hence ALUSrc=1, ForwardA = 01 and ForwardB=10 to select the result in the EX/MEM pipeline register generated by the ori $6,$0,4 instruction to pass on to the next stage in place of the stale value that was read for $6 when the sw was in the decode stage.

c) (3) When sw $8,8($8) is in the execute stage, show what the following bit patterns should be for the datapath shown above:
ForwardA = __00__, ForwardB = __00__, ALUSrc = __1___
The sw $8,8($8) instruction uses $8 as the upper ALU input and the constant 8 as the lower ALU input. Register $8 will have already been written by the sll instruction when sw $8,8($8) reads register $8, so no forwarding is required. So ForwardA = 00, ForwardB = 00, and ALUSrc=1 to select the constant 8 as the lower ALU input.

d) (3) When lw $6,0($8) is in the execute stage, show what the following bit patterns should be for the datapath shown above:
ForwardA = __00__, ForwardB = __00_, ALUSrc = __1__
The lw $6,0($8) uses the constant 0 as the lower ALU B-input, and $8 produced by the sll instruction as the upper ALU A-input. Note that a store word instruction (sw) writes to memory, it does not write $8. So no forwarding is required in this case either. ForwardA = 00, ForwardB = 00 and ALUSrc=1 to select the constant 0 as the lower ALU B-input. This lw instruction reads $6 in stage 2 even though it does not use $6 as an input operand. Register $6 would have already been written by the second ori instruction in any case.

11. The following instruction sequence is executed in order on the pipelined datapath:

Ins1:	or $5,$0,$0
Ins2:	lui $7,0x3A
Ins3:	addi $8,$7,0x4004
Ins4:	sw $5,24($8)
Ins5:	lw $8,44($5)
Ins6:	add $6,$8,$5
Ins7:	sw $5,64($5)

a) (6) Use the labels (Ins1, Ins2, etc.) to identify all instructions in the sequence for which there is a data hazard.
Ins3 (addi) requires $7 from Ins2 (lui) which is a data hazard.
Ins4 (sw) requires $8 from Ins3 (addi) which is a data hazard.
Ins6 (add) requires $8 from Ins5 (lw) which is a data hazard.

Ins4 (sw) depends on $5 from Ins1 (or). However, this is not a data hazard since Ins1 writes $5 before Ins4 reads $5.

b) (4) Use the labels (Ins1, Ins2, etc.) to identify all dependent instructions in the sequence for which data forwarding eliminates its data hazard.
Data forwarding eliminates the data hazard for Ins3 and Ins4.

c) (3) For any dependent instructions in the sequence for which data forwarding does not eliminate all stalls, identify the dependent instruction and state how many cycles the dependent instruction must be stalled.
Ins6 (add) must be stalled for one cycle to allow Ins5 (lw) time to complete its memory read stage.

Computer Science 605.611

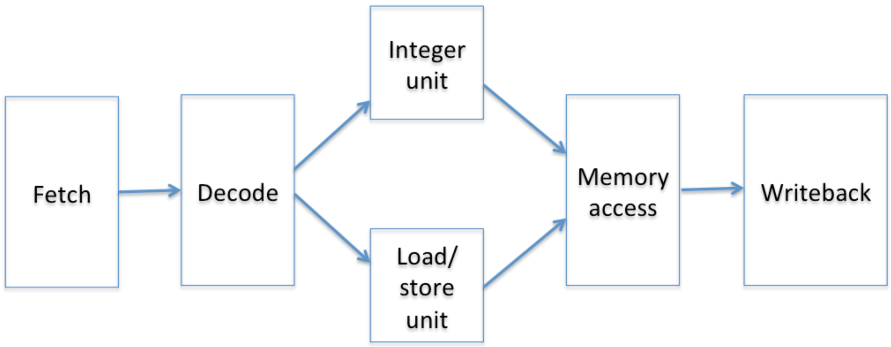
Problem Set 8 Answers

1. (5) Recall that our 5-stage MIPS pipeline is an example of a "scalar" pipeline since at most one instruction can occupy any stage at one time. Which of the following types of hazards does this pipeline never experience and why?

a) Data hazard
b) resource hazard
c) branch hazard
d) control hazard

A resource hazard would never occur, since at most one instruction would use the same resource at the same time.

2. The diagram below represents a degree 2 superscalar version of our 5-stage MIPS pipelined system. The decode stage includes a buffer to hold instructions that have been decoded but cannot yet execute. The execute stage includes an integer unit and a load/store unit. The system allows both out-of-order execution and out-of-order completion. Each stage requires 1 clock cycle.



a) (5) How many clock cycles does the following short instruction sequence consume on this superscalar system?

1. add $5,$4,$3
2. lw $2,16($6)
3. sw $9,8($7)
4. lw $10,48($12)
5.

After the first two instructions are fetched and decoded, they can execute in parallel on the two execution units. However the final sw and lw must execute sequentially since they both require the load/store unit. Hence the first two instructions complete in cycle 5, the sw completes in cycle 6 and the lw completes in cycle 7.

b) (3) How many clock cycles does the same instruction sequence consume on the original 5-stage scalar system?
On the original scalar system, the sequence would require 8 cycles. The add completes in cycle 5 with one additional cycle required for each of the remaining 3 instructions.

c) (3) How many read ports are required for the register file to support the superscalar system? Recall that a separate read port is required for each register input operand.
Since this is a degree 2 superscalar system, up to 2 instructions may have to read a pair of input registers at the same time, so 4 register read ports are required.

d) (3) How many write ports are required for the register file to support the superscalar system? Write ports receive the register results produced by instructions.
Two register write ports are required, one for each of the two instructions that can complete in a single cycle.

3. Consider the following instruction sequence that executes on our 5-stage scalar pipelined system with no code rearrangement and no branch prediction, but with a hazard unit as well as a forwarding unit:

```
        ori   $4,$0,2
loop:   add   $6,$7,$8
        and   $9,$4,$7
        bne   $4,$0,loop        ; delayed branch
        srl   $4,$4,1
        addi  $3,$0,1
        sll   $3,$3,1
```

a) (5) The ori instruction is fetched in cycle 1. During which clock cycle does the sll instruction complete the write-back stage if no early branch condition evaluation or branch prediction is used?

The ori instruction initializes $4 to the value 2. Each time that the bne instruction is executed, the srl in the delay slot is executed whether the branch is taken or not. Register $4=2 the first time through the loop, $4=1 the second time, $4=0 the final (third) time through the loop.
The bne takes effect when it is in the memory stage. When the bne branches, the instructions in the decode and fetch stages will be flushed causing two bubbles. The third time through the loop, no flushing occurs since the branch is not taken. The sll completes its write-back stage in cycle 23 since the instructions flow through the pipeline as follows:

Cycle	Fetch	Decode	Exec	Mem	Write-back
1	ori				
2	add	ori			
3	and	add	ori		
4	bne	and	add	ori	
5	srl	bne	and	add	ori
6	addi	srl	bne	and	add
7	sll	addi	srl	bne	and
8	add	bubble	bubble	srl	bne
9	and	add	bubble	bubble	srl
10	bne	and	add	bubble	bubble
11	srl	bne	and	add	bubble
12	addi	srl	bne	and	add
13	sll	addi	srl	bne	and
14	add	bubble	bubble	srl	bne
15	and	add	bubble	bubble	srl
16	bne	and	add	bubble	bubble
17	srl	bne	and	add	bubble
18	addi	srl	bne	and	add
19	sll	addi	srl	bne	and
20		sll	addi	srl	bne
21			sll	addi	srl
22				sll	addi
23					sll

b) (5) The ori instruction is fetched in cycle 1. There is no early branch condition evaluation. During which clock cycle does the sll instruction complete the write-back stage if branch prediction based on a decode history table (DHT) is used? Assume that the DHT always predicts that the branch will be taken.

When the branch is predicted in stage 2, the instruction in the delay slot (srl) will already be in the fetch stage. The next instruction that is fetched into the pipeline behind the srl instruction will be from the branch target address (i.e., the add instruction). The first two predictions will be correct, but the third is incorrect. When the bne actually branches, no flushing occurs. Flushing only occurs when the prediction is wrong. The incorrect branch prediction will be detected and acted on when the bne is in the MEM stage, which causes the two instructions (add, and) along the incorrect path to be flushed. Hence the instructions flow through the pipeline as follows:

Cycle	Fetch	Decode	Exec	Mem	Write-back
1	ori				
2	add	ori			
3	and	add	ori		
4	bne	and	add	ori	
5	srl	bne	and	add	ori
6	add	srl	bne	and	add
7	and	add	srl	bne	and
8	bne	and	add	srl	bne
9	srl	bne	and	add	srl
10	add	srl	bne	and	add
11	and	add	srl	bne	and
12	bne	and	add	srl	bne
13	srl	bne	and	add	srl
14	add	srl	bne	and	add
15	and	add	srl	bne	and
16	addi	bubble	bubble	srl	bne
17	sll	addi	bubble	bubble	srl
18		sll	addi	bubble	bubble
19			sll	addi	bubble
20				sll	addi
21					sll

Therefore the sll completes its write-back stage in cycle 21.

4. (5) Our MIPS system treats all branch instructions as delayed branches. Which one of the following is the main reason for using delayed branches?
a) to reduce the number of data hazards
b) to provide time to compute the branch target address
c) to reduce the number of pipeline bubbles
d) to provide time to predict the branch behavior
The main reason is to reduce the number of pipeline bubbles that result from flushing instructions that should not execute from the pipeline. With a decode history table the branch prediction occurs in stage 2 but the actual branch behavior is not known until stage 4 after the condition has been tested. When the branch instruction is in stage 4 there are 3 instructions behind it in the pipeline. Due to the delayed branch, only the first 2 stages have to be flushed resulting in a maximum branch penalty of 2 cycles.

5. a) (3) Once a conditional branch instruction is brought into the pipeline, what is the earliest stage within the pipeline in which a branch prediction can be made if the prediction is based on a branch history table?
Since only the address in the PC is required, the prediction can be made as early as the fetch stage (stage 1). There is no stage 0.

b) (3) When the prediction for a conditional branch instruction is found in a decode history table, what is the maximum branch penalty (i.e., the maximum number of pipeline bubbles that must be inserted) if the prediction turns out to be incorrect and the instruction is a delayed branch.
With a decode history table the branch prediction occurs in stage 2 but the actual branch behavior is not known until stage 4 after the condition has been tested. When the branch instruction is in stage 4 there are 3 instructions behind it in the pipeline. Due to the delayed branch, only the first 2 stages have to be flushed (in the fetch and in the decode stage) resulting in a maximum branch penalty of 2 cycles.

6. Consider the following code containing nested loops:

```
Loop1:  …
        …
Loop2:  …
        …
        …
        bne   $t2,Loop2
        …
        …
        beq   $t3,Loop1
        …
```

The bne and beq instructions are the only branch instructions within the code sequence. The outer loop (Loop1) contains an inner loop (Loop2). Assume that the beq at the end of the outer loop transfers control 120 consecutive times before the loop is exited. Also assume that for each iteration of the outer loop, the bne instruction at the end of the inner loop transfers control 100 consecutive times before the inner loop exits. That is, the inner loop iterates 100 times for each of the 120 iterations of the outer loop. Branch prediction is used for both branch instructions. What is the total number of mispredictions for each branch instruction, if:

a) the prediction is based on a separate single branch prediction bit for each branch instruction?

(3) Number of mispredictions for the bne = ___240_____

A misprediction occurs the first time the bne is executed and the bit is changed to 1 to indicate taken. So for each of the next 98 executions the prediction is correct. On the 100th execution the branch is not taken, and a second misprediction occurs. Hence there are 2 mispredictions for the bne for each of the 100 iterations of the inner loop (on the first and 100th). But the inner loop iterates 100 times for each iteration of the outer loop. So the total number of mispredictions for the bne = 2*120 = 240.

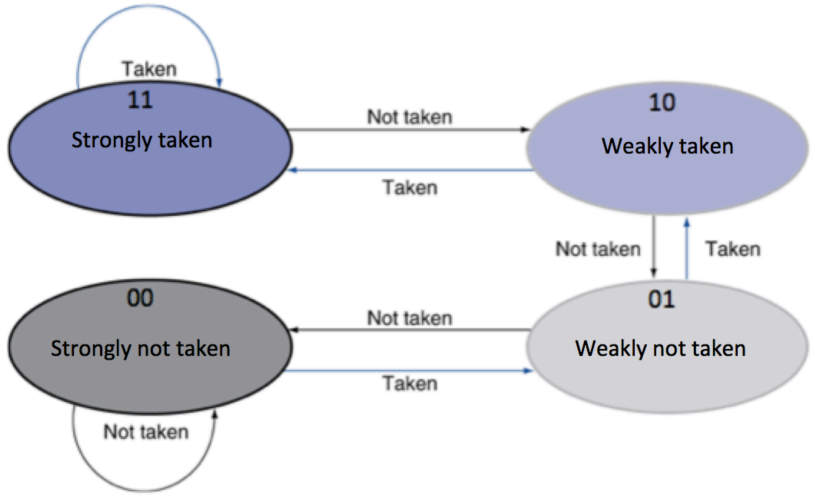
(3) Number of mispredictions for the beq = ____2_____

A misprediction occurs on the first execution of the beq instruction and on the final execution of the beq instruction (i.e., on the 120th execution).

So the total number of mispredictions for the beq = 2.

The initial value for the branch prediction bit for each branch instruction is 0 (not taken) and the bit is inverted each time that the prediction is incorrect (i.e., mispredicted).

b) the prediction is instead based on a separate pair of branch prediction bits for each branch instruction. The initial value of the branch prediction bits for each branch instruction = 00 (strongly not taken) and the bits are updated based on the diagram below:



(5) Number of mispredictions for the bne = ___122__

A misprediction occurs for the first execution of the bne instruction and the bits change from 00 to 01. On the second execution of the bne another misprediction occurs and the bits change from 01 to 10. The prediction is now for the bne to be taken so on the third execution of the bne the prediction is correct and the bits change from 10 to 11 (strongly taken). Hence for the next 96 executions of the bne the prediction will be correct. On the 100th iteration there is a misprediction and the bits go from 11 back to 10 (weakly taken). Hence for the first iteration of the outer loop, the bne at the end of the inner loop is misprediction three times. But for the remaining iterations of the outer loop, the first 99 predictions for the bne at the end of the inner loop will be correct and only the final iteration of the inner loop will have a misprediction. This is because the prediction bits for the bne are left at 10 (taken) each time the inner loop completes. So the total number of mispredictions for the bne = 3 + 119 = 122.

(5) Number of mispredictions for the beq = _____3___ There is a misprediction for the beq instruction on the first and second iteration of the outer loop. On the first misprediction the bits change from 00 to 01. On the second misprediction the bits change from 01 to 10 (weakly taken). On the third iteration the prediction is correct and the bits change from 10 to 11 where they remain for the next 116 iterations. On the 120th iteration there is a misprediction but the bits go from 11 back to 10. So the total number of mispredictions for the beq instruction is 3.

7. a) (3) What type of data hazard (RAW, WAR, WAW, or NONE), requiring a stall, is caused by the following pair of instructions on a scalar pipeline?

```
sw    $5,44($2)
add   $10,$5,$2
```

There is no data hazard since the sw instruction does not write to $5. So the answer is NONE.

b) (3) What type of data hazard (RAW, WAR, WAW, or NONE), requiring a stall, is caused by the following pair of instructions on a scalar pipeline?

```
sw    $5,44($2)
lw    $5,44($2)
```

None. The sw does not modify $5 and the lw does not use $5 as an input. Since instructions execute in-order on the scalar pipeline, the sw completes its write to memory before the lw reads from the location. Also since the instructions execute in the order listed, the lw can't overwrite $5 before the sw reads $5

c) (3) What type of data hazard (RAW, WAR, WAW, or NONE) can be caused by the following two instructions if instead of the scalar system they are executed on a superscalar pipeline with multiple load/store units that allows out-of-order execution?

```
sw    $5,44($2)
lw    $5,44($2)
```

On a superscalar system the lw could be allowed to execute before the sw (for example, if the sw was stalled waiting on a result in $5). If so, the lw would change the value in $5 that the sw would write to memory. This would be a WAR data hazard on register $5.

Also a RAW data hazard exists on the shared memory word since the lw must read from memory after the sw writes the same memory word. Otherwise the lw would read the previous contents of the memory word.

8. (10) On a certain VLIW system, each long instruction word is a packet or bundle of six individual machine instructions. The hardware system contains three integer units and three floating point units. Therefore, up to 3 integer operations and 3 floating point operations can be performed in parallel in the same clock cycle. However, due to the mix of available instructions and possible dependencies, the system may not be able to make use of all the execution units in every cycle. In that case, one or more nop instructions must be inserted into the instruction bundle (i.e., into the long instruction word) to fill any unused slots.
Using the following format to show the instructions within each long instruction word:

Int unit1	Int unit2	Int unit3	Flt unit1	Flt unit2	Flt unit3

List the contents of the all long instruction words or bundles that are required for the following group of instructions using the minimum number of nop instructions:

```
add     $11,$2,$3
add     $4,$5,$11
mtc1    $4,$f12
cvt.s.w $f12,$f12
sub.s   $f14,$f16,$f18
add.s   $f8,$f14,$f12
sll     $6,$9,5
sub.s   $f8,$f14,$f8
```

Based on the available execution units and the dependencies among the instructions, the following 6 long instruction words will be produced:

Int unit1	Int unit2	Int unit3	Flt unit1	Flt unit2	Flt unit3
add $11,$2,$3	sll $6,$9,5	nop	sub.s $f14,$f16,$f18	nop	nop
add $4,$5,$11	nop	nop	nop	nop	nop
mtc1 $4,$f12	nop	nop	nop	nop	nop
nop	nop	nop	cvt.s.w $f12,$f12	nop	nop
nop	nop	nop	add.s $f8,$f14,$f12	nop	nop
nop	nop	nop	sub.s $f8,$f14,$f8	nop	nop

Floating point instructions must execute on a floating point unit and CPU instructions (integer instructions) must execute on an integer unit. The mtc1 instruction is a CPU instruction.
The add must write $4 before the mtc1 reads $4. The mtc1 most copy $4 into $f12 before the cvt.s.w can convert the contents of $f12. The cvt.s.w must write $f12 before the add.s reads $f12 and the add.s must write $f8 before the sub.s reads $f8.

9. (5) What is the main advantage of using separate reservation stations versus using a centralized instruction window to supply instructions to the execution units within a superscalar system?

When a reservation station becomes full, only later instructions that require the same reservation station are stalled. However, a full instruction window causes later instructions of all types to be stalled. So there is a greater chance that one or more execution units will be idle with a full instruction window. One or more full reservation stations still allow further instructions to be processed as long as those instructions map to execution units whose reservation station still has room. The instruction window can allow a single instruction type to monopolize the window and starve some execution units for work.

10. (5) Which type of data hazard (RAW, WAR or WAW) is best handled by using a reorder buffer (ROB)? Explain why.

The answer is WAW, since instructions are extracted from the ROB in the Writeback stage, so that the result registers are written in the correct program order.

11. (5) For each of the 3 types of data hazard: RAW, WAR and WAW, indicate whether register renaming can or can not be used to avoid stalling the pipeline and explain why.

Register renaming can be used to handle both WAR and WAW hazards, but not RAW hazards.

12. a) (5) What is the minimum width (in bits) required for the CPU-to-instruction memory bus in a MIPS degree 4 superscalar system?

The system must be able to fetch four instructions at a time. So the bus must be at least 4*32 = 128 bits wide.

b) (5) Does hardware or does software select the instructions that execute together in the same clock cycle on a:

i. VLIW system

The compiler selects the instructions that it combines into a VLIW instruction or bundle as it translates the source program. So the answer is software.

ii. superscalar system.

The hardware controller selects, at runtime, the instructions that execute together during the same cycle.

Computer Science 605.611

Problem Set 9 Answers

(Recall that by default the MIPS system uses big endian storage order for memory)

1. The addresses and 8-bit contents of four consecutive memory bytes on a system that uses big endian storage order are shown below:

Address	Contents
0x10040006	0xF1
0x10040007	0xF2
0x10040008	0xF3
0x10040009	0xF4

Register $6 contains the 32-bit address 0x10040006.

a) (3) Show (in hex 0xdddddddd) the 32-bit contents of register $2 after executing the instruction:

lh $2,2($6) $2 = _____0xFFFFF3F4___

This instruction loads a halfword from address 2+0x10040006 = 0x10040008 into the low half of register $2. The sign bit is extended throughout the upper 16 bits of $2.

b) (3) Show (in hex 0xdddddddd) the 32-bit contents of register $3 after executing the instruction:

lhu $3,0($6) $3=_____0x0000F1F2___

This instruction loads a halfword from address 0x10040006 into the low half of register $2 as an unsigned 16-bit value. The upper 16 bits of $2 are all set to 0.

2. An integer array that contains two 32-bit elements is declared as:
unsigned integer idata[2];
The element idata[0] contains the value 0x07176305
The element idata[1] contains the value 0x60151930

The memory address of element idata[0] is 0x100900C0. The address of idata[1] is 0x100900C4. Indicate in hex, the contents of the single 8-bit memory byte located at each address listed in the tables below for:

a) (8) a little endian memory system

Address	8-bit Contents
0x100900C1	0x63
0x100900C4	0x30

With little endian storage order, the bytes within a multi-byte item are ordered from low byte to high byte. The address of the low byte of idata[0] is 0x100900C0, the high byte is at address 0x100900C3. The address of the low byte of idata[1] is 0x100900C4, the high byte of idata[1] is at address 0x100900C7.
The bytes within the two elements appear in memory in the following order:

Address	8-bit Contents
0x100900C0	0x05
0x100900C1	0x63
0x100900C2	0x17
0x100900C3	0x07
0x100900C4	0x30
0x100900C5	0x19
0x100900C6	0x15
0x100900C7	0x60

b) (8) a big endian memory system

Address	8-bit Contents
0x100900C2	0x63
0x100900C6	0x19

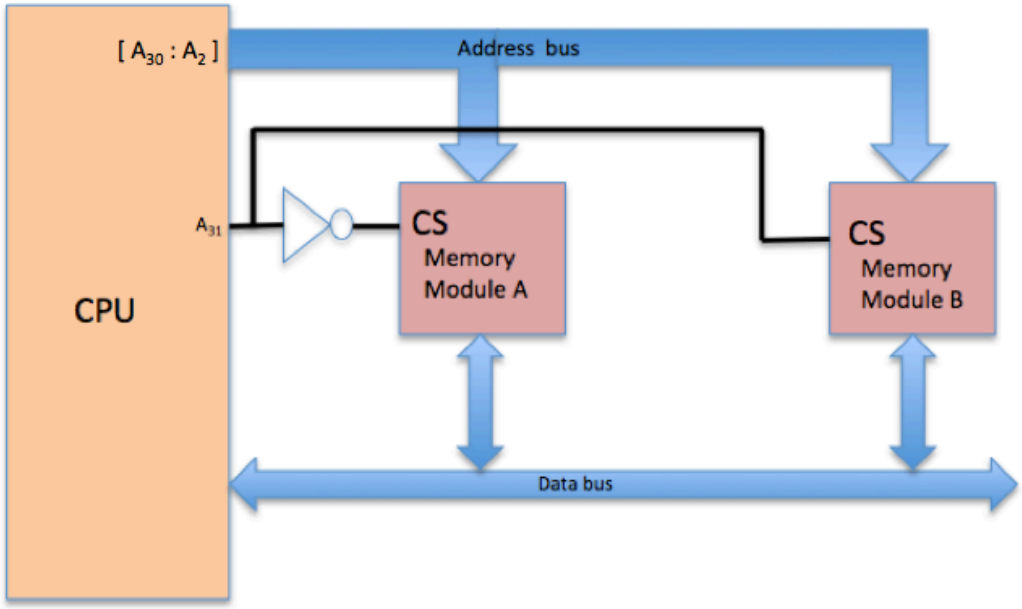
With big endian storage order, the bytes within a multi-byte item are ordered from high byte to low byte. The address of each 32-bit element is the address of the high byte within the element. The address of the high byte of idata[0] is 0x100900C0, the low byte is at address 0x100900C3. The address of the high byte of idata[1] is 0x100900C4, the low byte of idata[1] is at address 0x100900C7. The bytes within the two elements appear in memory in the following order:

Address	8-bit Contents
0x100900C0	0x07
0x100900C1	0x17
0x100900C2	0x63
0x100900C3	0x05
0x100900C4	0x60
0x100900C5	0x15
0x100900C6	0x19
0x100900C7	0x30

(The bytes within a register are always ordered from high byte on the left to low byte on the right.)

3. The memory system shown below consists of two modules (A and B). Each module has a depth of 536870912 and a width of 32. That is, each cell is 32 bits wide and there are 536870912 cells in each module. The memory storage cells within each module

are numbered starting from 0. The system employs 32-bit addresses (A31 – A0). However since each memory cell is 4 bytes wide, the 2 low bits of the memory address for each cell are always 00. So bits A1 and A0 are not used on the address bus.



The chip select (CS) is active high. That is, to select a chip, its CS input must =1.

a) (3) What is indicated by the bit pair A0 and A1 within the 32-bit memory address on this system?
These 2 bits are used as an offset to select a byte within the cell after it has been read.

b) (3) What is the minimum number of bytes transferred from memory to the CPU for a single read operation?
Each read obtains the contents of a single memory cell, which is four bytes (32 bits).

c) (3) What is the total number of bytes contained in module B?

Since the module contains 536870912 cells and each cell is 4 bytes, the total number of bytes in the module = 4 * 536870912 = 2147483648.

(3) What is the 32-bit hex address of cell 7 (the 8th cell) within module A?

A_{31} must be 0 (which is inverted and used as the chip select), bits A_{30} through A_2 must contain 7. The 32-bit address = 00000000000000000000000000011100 = 0x0000001C.

Since each cell is 4 bytes, the low two bits of the 32-bit address for each cell will always be 00.

e) (3) What is the 32-bit hex address of the next to last cell within module B?

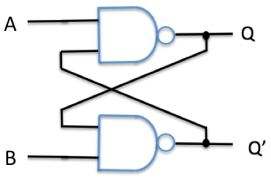
The cells are numbered 0 through 536870911 or in hex 0x0 to 0x1FFFFFFF. The next to last cell is cell 536870910 = 0x1FFFFFFE. This is the value that must be contained in the 29 bits (A_2 through A_{30}). A_{31} must be 1 to select module B. A_1 and A_0 will both be 0.
Therefore the 32-bit address of the next to last cell in module B in binary is 11111111111111111111111111111000 = 0xFFFFFFF8.

f) (4) If register $4 contains 0x80000000, which module and which byte within the module (byte 0 is the 1st, byte 1 is the 2nd byte, etc) is referenced by the instruction:

lbu $6,5($4)

The memory byte referenced is at address 0x80000005. Bit 31 = 1 so module B is selected. Bits 2 through 30 indicate cell 1 within module B. The two rightmost bits = 01 correspond to the second byte within the cell which is the 6th byte within the module.

4. (8) Complete the characteristic table for the circuit shown below by filling in the column for Q(t+1), the output in the next cycle, given that Q(t) is the output in the current cycle and the digital inputs in the current cycle are A and B. If there is no change then Q(t+1)=Q(t), otherwise Q(t+1) = 0 or Q(t+1) = 1.



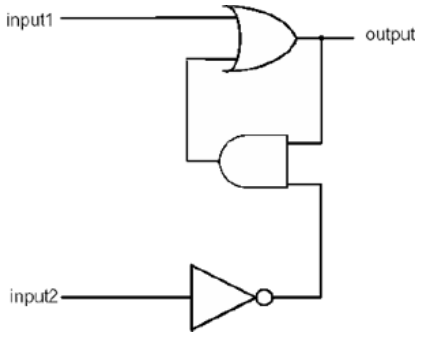
A	B	Q(t+1)
0	0	?
0	1	1
1	0	0
1	1	Q(t)

The cross coupled NAND gates within the circuit form an SR latch with active low inputs. Recall that (1 NAND X) = X', the complement of X. So if both A and B are 1, the upper NAND gate outputs the complement of Q' which is Q and the lower NAND gates outputs the complement of Q which is Q' (so the outputs retain there previous values). This means the A and B are de-asserted when each is 1 and are asserted when 0 (i.e., they are active low).

When both A and B are asserted (i.e., set to 0) each NAND gates outputs 1 (since 0 NAND X = 1). However, when A and B are de-asserted again, the final value of Q depends on whether A or B reaches 1 first. So the final output is unpredictable.
Asserting only A causes the output Q to go to 1 which then forces Q' to 0. Then de-asserting A latches or retains the current value for Q (=1).

Asserting only B causes the Q' to go to 1 which then forces Q to 0. Then de-asserting B retains the current output values (Q=0, Q'=1).

5. (5) Which one of the following (a. through e.) best describes the circuit below? Explain your answer.



a. D Latch
b. D Flip-flop
c. S-R Latch
d. S-R Flip-flop
e. Comparator

A clock does not control when the output changes so this is not a flip flop.

In the zero state output=0 by definition. In the 1 state output=1. For state 0, if both input1 and input2 are 0, the AND gate generates (output AND input2') = 0 AND 1 = 0, so the OR gate outputs 0 OR 0 = 0. Thus the zero state is retained.

In the zero state, if input1 = 1 and input2 = 0, the OR gate outputs 1 (the state changes to 1). The output remains 1 even when input1 is then de-asserted since the AND gate still produces (1 AND 1) = 1 [i.e.. output AND (input2)']. Thus the one state is retained.

In the one state, asserting input1 leaves the output at 1 but asserting input2 forces the output to 0. This is because the inverter outputs (input2)' = 0. This forces the AND gate to output 0 which causes the OR gate to generate output = 0 OR 0 = 0. So input1 acts like a SET signal and input2 acts like a RESET signal and the circuit behaves like an S-R latch.

If both input1 and input2 are asserted (i.e., set to 1), the AND gate outputs 0 and the OR gate outputs 1. When they are de-asserted to retain the output, the output stays at 1 if input2 reaches 0 first; but the OR gate outputs 0 if input1 reaches 0 first. So the output is unpredictable just like for an S-R latch.

The circuit is not a comparator since the output can be 0 or 1 when the inputs are equal or when they differ.

6. Recall that performing a read from a dynamic RAM (DRAM) requires that the chip pre-charge before it can supply the requested data. Assume that it takes 10ns to pre-charge and 15ns to either output the requested data in response to a read operation or to store the input data for a write operation. Also recall that our MIPS pipeline system employs a Harvard Architecture, transfers 32 bits at a time between the CPU and memory, and each pipeline stage consumes one clock cycle.

   a) (5) If the memory stage is the most time consuming stage, what is the maximum clock rate (expressed in GHz) that can be used for our 5-stage pipelined system using this type of DRAM memory?

The maximum clock rate would be determined by the longest stage (the memory stage). The cycle time must be at least as long as the memory cycle time = 10ns + 15ns = 25ns. This corresponds to a maximum clock rate of 1/25ns = $0.04 * 10^9$ Hz = 0.04 GHz.

   b) (5) The instruction memory contains one machine instruction per cell and is implemented as a single memory module with a storage capacity of 268435456 bytes, what is the width and depth of the module?

The width of each storage cell is 32 bits (4 bytes) which is the size of a machine instruction. The depth of the memory module is the number of cells in the module = $(256* 2^{20}) /4 = 64 * 2^{20} =$ 67108864.

7. The diagram below shows the 32-bit address format used for a particular byte addressable memory system containing 16 modules:

31 28	27 1	0
Module#	Cell number	offset within cell

a) (3) What is the width in bits of each memory module? Width = _16_ bits.
A single-bit offset corresponds to 2 bytes per cell (i.e., 2*8 = 16 bits)

b) (4) How many memory modules must be accessed to read a single 32-bit word from address 0x00408000 with this memory system?
Since the module number appears in the leftmost field, this is a high-order interleaved system with adjacent cells within the same module. So a single module has to be accessed twice to obtain a single 32-bit word from address 0x00408000.

c) (4) How many memory modules must be accessed to read a single 32-bit word from address 0x3FFFFFFE with this memory system?
This address corresponds to the final cell within module 3. So two modules must be accessed to obtain a single 32-bit word from this address (the final cell in module 3 and the first cell in module 4).

8. (9) Our byte addressable MIPS system employs a 32-bit CPU-to-memory bus as well as memory cells that are 32 bits wide. Recall that an aligned memory access is one for which the

memory address used for the read or write is a multiple of the size (in bytes) of the item that is being transferred.
Ignoring any exceptions that may occur, how many data memory cells must be read by each of the following instructions if the contents of register $8 = 0xB003D8E0 ?

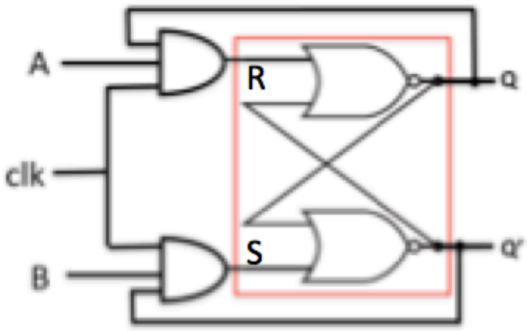
I. lb $t0,54($8)
Only one memory cell is read because the byte resides within a single 32-bit cell.

II. lh $t0,54($8)
Decimal 54 = 0x36 so the address is 0xB003D8E0+0x36 = 0xB003D916 which falls within the word at address 0xB003D914 (containing the 4 bytes at addresses 0xB003D914, 0xB003D915, 0xB003D916 & 0xB003D917 respectively). The two bytes at addresses 0xB003D916 and 0xB003D917 reside within the same cell, so only one cell is read.

III. lw $t0,54($8)

The address is 0xB003D8E0+0x36 = 0xB003D916. This address falls within the word whose bytes are at addresses 0xB003D914, 0xB003D915, 0xB003D916, and 0xB003D917. Hence the first 2 bytes are in this word and the next 2 bytes are in the next adjacent word whose bytes are at addresses 0xB003D918, 0xB003D919, 0xB003D91A and 0xB003D91B. Hence two cells must be read to obtain the required 4 bytes (the final 2 bytes of one cell and the first 2 bytes of the next cell). An unaligned exception occurs to signal the problem.

9. (8) Complete the characteristic table for the circuit shown below by filling in the column for Q(t+1), the output in the next cycle, given that Q(t) is the output in the current cycle and the digital inputs in the current cycle are A and B which take effect when the clock signal (clk) pulses from low to high and back to low. The width of the pulse is such that the outputs Q and Q' only change once. If there is no change then Q(t+1)=Q(t), otherwise Q(t+1) = 0 or Q(t+1) = 1.



A	B	Q(t+1)
0	0	Q(t)
0	1	1
1	0	0
1	1	Q(t)'

This circuit is just the NOR gate implementation of an S-R latch. But here the upper AND gate generates the RESET signal (R) and the lower AND gate generates the SET signal (S). Both AND gates output 0 when the clk is low so there is no change in the outputs Q and Q'.

When the clk signal goes from low to high, the output of the upper AND gate depends on A and Q; the output of the lower AND gate depends on B and Q'.

When A is 1 and Q is 1, the output of the upper AND gate (R) is 1 (a reset operation)
When B is 1 and Q' is 1, the output of the lower AND gate (S) is 1 (a set operation)

If A and B are both 1 at the same time, the upper AND gate outputs 1 AND Q = Q, and the lower AND gate outputs 1 AND Q' = Q'. So if Q is 1, R= 1 AND 1 = 1 (Q is reset). If Q=0 (i.e., Q' is 1), S= (1 AND 1) = 1 (Q is set). Hence Q(t+1) = Q(t)' when both A and B are 1 (the output toggles).

A circuit called a J-K flip flop behaves in this way. That is, at the clock edge J-K flip flop toggles the output if the inputs J and K are both 1, the output is unchanged if J and K are both 0, the output is reset to 0 if J=1 and K=0, the output is set to 1 if J=0 and K=1.

10. A computer system designed and built in the 1970's made use of a single 8-bit CPU-to-memory data bus and a single separate address bus. Suppose that such a system contained 4 memory modules or chips, each of which was 8 bits wide with a depth of 16777216. A memory access for each module consisted of only two phases: an addressing phase in which an address is sent to select a specific memory module and cell within the module; and a data transfer phase in which the selected data item is read from or written into. Each phase takes 40 nano-seconds to complete. What is the minimum time (in nano-seconds) required to read a single 32-bit data item from this memory system if it employs:

a) (4) high order interleaving

Since the data bus is only 8 bits wide, only a single byte can be transferred over the bus at one time.

With high order interleaving, consecutive or adjacent cells reside within the same module. So a separate addressing phase and data transfer phase is required for each byte. The minimum time required to read a single 32-bit data item is therefore 4*(40+40) = 320 ns.

b) (4) low order interleaving

With low order interleaving, consecutive cells reside within different modules.

Hence the modules can operate in parallel so that while a data transfer occurs for one module the addressing phase for another module can occur in parallel.

Therefore the first byte is obtained after 40+40 = 80 ns and the addressing phase for the second module overlaps with the data transfer phase of the first, etc. So the entire group of 4 cells required for the complete 32-bit data item is obtained in

80 + 40 + 40 + 40 = 200 ns.

Computer Science 605.611

Problem Set 10 Answers

(Recall that MIPS memory is byte addressable; its size is measured in units of bytes not bits.)

1. (5) A MIPS system employs a 4-way set associative I-cache (i.e., instruction cache) that can hold a total of 262144 instructions and uses a 256-byte line size. The second instruction within way3 of set 0x29 is fetched. If way3 has tag 0x345, what is the corresponding 32-bit physical address (expressed in hex 0xddddddddd) of the instruction that is fetched?

Since the cache holds 262144 four-byte instructions, its size = 4*262144 = 1048576 bytes. Each set contains 4 ways (i.e., lines), so the total number of sets in the cache is 1048576 / (4*256) = 1024. Therefore 10 bits are required for the set number. Each line is 256 bytes, so 8 bits are needed for the offset within a line.
Hence the tag field requires 32 - 10 - 8 = 14 bits.
The second instruction within each line is at offset 4. The 14-bit tag field contains the value 0x345, the 10-bit set field contains the value 0x29 and the 8-bit offset field contains the value 4.
This corresponds to address 00001101000101 0000101001 00000100 = 0x0D142904

2. (5) A fetch of an instruction from address 0x4048C824 on a MIPS systems causes a hit in the I-cache. The direct-mapped I-cache is 2097152 bytes in size and has 64-byte cache lines. Show, in decimal, the tag, the line number and the offset within the line for the instruction that is fetched. Tag = 514__, Line# = ___8992___, Offset = ___36__

The cache contains 2097152/64 = 32768 lines,
so the line number field requires $\log_2(32768) = 15$ bits.
The offset field width = $\log_2(64) = 6$ bits.
So the leftmost 32 – (15+6) = 11 bits contain the tag.
0x4048C824 = 01000000010 010001100100000 100100
Therefore the tag = 01000000010 = 0x202 = decimal 514
Line number = 010001100100000 = 0x2320 = decimal 8992
Offset = 100100 = 0x24 = decimal 36

3. The MIPS instruction sequence below shows one way to implement the high level statement: X = X+Y

```
lui     $t0,0x3D5A          ; load $t0 with base address 0x3D5A0000
lui     $t2,0x3D5A          ; load $t2 with the same base address
lw      $t1,0x4BFC($t0)     ; read variable X into $t1
lw      $t3,0x4B08($t2)     ; read variable Y into $t3
add     $t1,$t1,$t3         ; compute X + Y
sw      $t1, 0x4BFC($t0)    ; store sum back into X
```

The 32-bit variables X and Y reside at physical memory addresses 0x3D5A4BFC and 0x3D5A4B08 respectively.
If the corresponding machine code is executed on a system with a D-cache (data cache) that is a direct mapped, write allocate, copy-back cache of size 524288 bytes, how many D-cache hits and how many D-cache misses occur if the D-cache is initially empty and the line size is:

a) (5) 128 bytes
If the line size is 128 bytes, then the number of lines in the D-cache = 524288/128 = 4096, so the line number field contains 12 bits and the offset field is 7 bits.
X resides at address 0x3D5A4BFC = 00111101010110100100101111111100, which maps to line 010010010111= 0x497
Y resides at address 0x3D5A4B08 = 00111101010110100100101100001000 which maps to line 010010010110 = 0x496.
When X is first read a miss occurs and the block containing X is brought into line 0x497 of the D-cache. When Y is read a miss also occurs and the memory block containing Y is brought into line 0x496 of the cache. When the sum is written to X, a cache write hit occurs in line 0x497. So 2 read misses and 1 write hit occurs for the D-cache.
#read hits = _0__ #read misses = __2___
#write hits = _1__ #write misses = _0__

b) (5) 1024 bytes
For 1024-byte lines, the number of lines in the D-cache = 524288/1024 = 512, so the line number field contains 9 bits and the offset contains 10 bits.

X resides at address 0x3D5A4BFC = 00111101010110100100101111111100, which maps to line 010010010 = 0x92
Y resides at address 0x3D5A4B08 = 00111101010110100100101100001000 which also maps to line 010010010= 0x92.
Both X and Y reside in the same memory block. Recall that the memory block size and the cache line size are always the same.
When X is first read a miss occurs and the block containing X is brought into line 0x92 of the D-cache. When Y is read, a hit occurs since X and Y are within the same cache line. When the sum is written, a cache write hit occurs.
So 1 read miss and 1 read hit occurs in the D-cache. One write hit and zero write misses occur in the D-cache.
#read hits = _1__ #read misses = __1___
#write hits = _1__ #write misses = _0__

4. a) (5) Assume that our MIPS system employs a 2-way set associative D-cache that has a total data storage capacity of 4194304 bytes and a 32-byte line size. To which set does address 0xC0404248 map?

For a 2-way set associative caches, each set contains two lines so the set size is 2 * 32 bytes = 64 bytes.

Hence the number of sets in the cache is 4194304/64 = 65636, so 16 bits are required for the set number. For 32-byte lines the offset can range from 0 to 31 and requires 5 bits.

0xC0404248 = 11000000010 0000001000010010 01000 so the address maps to set 0000001000010010 binary = 0x0212 = decimal 530.

b) (5) Assume instead that the D-cache has a total data storage capacity of 8388608 bytes and a 4-way set associative organization with a 64-byte line size. To which set does address 0x4248C040 map?

Each set is 4 * 64 bytes = 256 bytes in size.

Hence the number of sets in the cache is 8388608/256 = 32768, so 15 bits are required for the set number. For 64-byte lines, the offset can range from 0 to 63 and requires 6 bits.

0x4248C040 = 01000010010 010001100000001 000000 so the address maps to set 010001100000001 binary = 0x2301 = decimal 8961.

5. Recall that with a write-allocate policy, if a cache miss occurs the memory block containing the referenced address is first loaded into the cache and then updated. Suppose that for a 4-way set associative cache, the 3 pseudo-LRU bits for set 7 are currently 100 and the set is full. The cache employs a write-allocate policy and the next memory reference that maps to set 7 is a write miss. The pseudo-LRU bits are described in the sub-module on cache replacement policies.

a) (10) Which of the 4 lines (way0, way1, way2 or way3) within set 7 will be replaced in response to the write miss?

Due to the write-allocate policy, a new line is brought into set 7 in response to the miss. The pseudo-LRU bit pattern 100 selects way 0 to be replaced.

B2 B1 B0	Way to replace
000	0
001	2
010	1
011	2
100	0
101	3
110	1
111	3

b) (5) Suppose that the reference to set 7 had instead been a read hit in way2, and that the three pseudo-LRU bits for set 7 were 010 before the read was performed. After the read hit, what is the pattern for the three pseudo-LRU bits?

Since way 2 was accessed, B2 is set to 1 and B0 is set to 0. Hence the 3 bits are changed from 010 to 110.

Way accessed	Effect on LRU bits
0	1->B1, 1->B0
1	0->B1, 1->B0
2	1->B2, 0->B0
3	0->B2, 0->B0

Reading, writing or replacing a line is considered an "access"

LRU bits are updated for each access

6. A system employs a unified L1 cache with a read access time of 8ns. The main memory has a read access time of 60ns.

a) (6) Using the decimal format dd.dd% (i.e., 2 digits before and 2 digits after the decimal point) for your answer, what read hit ratio is required to produce an average read access time of 14ns if the cache operates in look-through mode.

Read hit ratio = ___90.00%_

For look-through mode, the average access time is given by

ta = tc + (1-h)*tm so (1-h) = (ta – tc)/tm

h = 1 - (ta – tc)/tm = 1 - (14-8)/60 = 54/60 = 0.90 or 90%

b) (6) If instead, the cache operates in look-aside mode and has a read hit ratio of 93%, what is the average read access time in nano-seconds?

Average read access time = __11.64 ns_

For look-aside mode, the average access time is given by

ta = h*tc + (1-h)*tm = 0.93*8 + 0.07*60 = 7.44 + 4.20 = 11.64 ns.

7. A virtual memory system employs 45-bit virtual addresses, uses a page size of 8192 bytes and contains 268435456 words of physical memory (each word, as usual, is 32 bits).

a) (4) State the minimum number of bits required for physical addresses on this system.
There are 268435456*4 = 1073741824 = 2^{30} bytes of physical memory. So at least 30 bits are needed for the physical address.

b) (5) What is the minimum size (in bytes) of the inverted page map table that could be used for this system if each entry in the inverted page map table contains only a valid bit, a dirty bit, a 6-bit task id plus any required addressing bits.
In addition to the valid bit, the dirty bit, and the 6-bit task id, each page table entry (PTE) must also contain the page number of the page that occupies the corresponding frame. The 45-bit virtual address contains a 13-bit offset for the 8192-byte pages. Hence the number of bits in the virtual page number is the virtual address size minus the offset size = 45-13 = 32.
Therefore each PTE must contain at least a 32-bit page number, a valid bit, a dirty bit and a 6-bit task ID for a total of 40 bits. Five bytes are required to hold all 40 bits. There is a separate PTE for each frame and there are 1073741824 ÷ 8192 = 131072 frames. So the inverted page map table must be at least 131072*5 bytes = 655360 bytes in size.

8. A matrix of 32-bit integers, x[512][32] (i.e. 512 rows with 32 elements per row) resides at memory address 3EA4000 on a machine with a virtual memory system that employs a page size of 8192 bytes. Assume that the amount of physical memory available to contain pages from the matrix is 32768 bytes, and that an LRU page replacement algorithm is used. Initially, all page map table entries are invalid (i.e., none of the required pages are in memory). Considering only the page faults generated in referencing the matrix (i.e., ignoring those that might result from instruction fetches or referencing the loop indices), indicate how many page faults would occur in executing the C language instruction sequence shown below. Note that in the C language, matrices are stored in row major order (i.e., one row after another).

a) (10)
```
for (j = 0; j < 30; j++) {
    for (k = 0; k < 350; k++)    x[k][j]  = x[k][j] + 1;
}
```
Number of page faults due to accessing the matrix = __180_.
Explain how you arrived at your answer.

There are **32768 /8192** = 4 frames of physical memory available. For each of the 30 columns (0 through 29) this code accesses rows 0 through 349 in sequence. That is, consecutive accesses hop from one row to the next. Each row is 32*4=128 bytes in size.

So each page can hold 8192/128 = 64 rows. The code accesses rows 0 through 349 causing page faults as indicated in the following table:

Row access causing page fault	rows loaded in response to page fault	Frame used
0	0 – 63	0
64	64 – 127	1
128	128 – 191	2
192	192 – 255	3
256	256 – 319	0
320	320 – 351	1

There are 6 page faults from the first iteration of the inner loop. Due to the LRU page replacement, each page will be replaced before it is required a second time. Therefore, each iteration of the inner loop will cause 6 page faults. The outer loop is executed 30 times, so the total number of page faults is 30*6 = 180.

b) (10) Suppose that instead, the system employs 4096-byte pages and has a total of eight 4096-byte frames of physical memory. An LRU page replacement algorithm is used and all page map table entries are initially set to invalid. Considering only the page faults generated in referencing the matrix (i.e. ignoring those that might result from instruction fetches or the loop indices), indicate how many page faults occur in executing the instruction sequence shown below using row major storage order for the matrix.

```
k=0;
while( k < 350 ) {
        j = 0;
        while( j < 30)   {
                x[k][j]  = x[k][j] + 1;
                j = j + 2;
        }
   k = k + 2;
 }
```

Number of page faults due to accessing the matrix = 11_

Explain how you arrived at your answer.

Each page contains 4096/128 = 32 rows. This code accesses even numbered rows 0 through 348 of the matrix (125 rows). Within

each row the 15 even numbered columns 0 through 28 are accessed in sequence. Hence after a page is loaded in response to a fault, the inner loop generates no additional page faults since the references made in the inner loop remain on the page. The code causes page faults as indicated in the following table:

Row access causing page fault	rows loaded in response to page fault	Frame used
0	0 – 31	0
32	32 – 63	1
64	64 – 95	2
96	96 – 127	3
128	128 – 159	4
160	160 – 191	5
192	192 - 223	6
224	224 - 255	7
256	256 - 287	0
288	288 - 319	1
320	320 - 351	2

Each page fault brings in 32 rows, so all 350 rows will have been loaded after 11 page faults. Hence only 11 pages faults are caused by this code.

9. A computer uses a byte-addressable virtual memory system with a TLB (translation look-aside buffer) that contains four entries, a 2-way set associative cache, and a page map table for a process P. Cache blocks are 8 bytes in size, while pages are 16 bytes in size. Assume, for the sake of this problem, that main memory contains only 4 frames and that the TLB and page table contents for Process P are as shown below. There is a separate PTE (page table entry) for each of the pages in the virtual address space. Based on the information in these tables, answer the following questions.

TLB	
page	frame
0	3
4	1
–	–
–	–

	Page Table	
	frame	Valid
0	3	1
1	0	1
2	-	0
3	2	1
4	1	1
5	-	0
6	-	0
7	-	0

a) (4) What is the minimum number of bits required for a virtual address?

Page map tables contain a separate PTE for each virtual page. Since there are 8 entries in the page map table, there are 8 virtual pages, so the page number requires 3 bits. A 16-byte page requires a 4-bit offset. So the minimum number of bits needed for the virtual address is 3+4 =7.

b) (4) What is the minimum number of bits required for a physical address?

The problem states that there are four frames, so two bits are required for the frame number. Frames are the same size as pages, so each frame is 16 bytes in size, which requires a 4-bit offset. The minimum number of bits in the physical address is 2+4 = 6.

c) (3) If the virtual address 0x48 is referenced, does a TLB hit occur?

Virtual address 0x48 corresponds to page 4. The second entry in the TLB indicates that page 4 resides in frame 1, so a TLB hit occurs.

d) (3) Does referencing virtual address 0x34 cause a TLB hit? To what physical address (if any) does virtual address 0x34 correspond?

0x34 = 0110100 as a 7-bit binary number

So the page number is 3 (binary 011) and the offset is 4 (binary 0100).

There is no entry in the TLB for page 3, so a TLB hit does not occur. Checking the PMT, we see that page 3 maps to frame 2. So virtual address 0x34 maps to physical address 100100 binary = 0x24.

Computer Science 605.611

Problem Set 11 Answers

1. The read access delay for an I/O system is defined as the time required for the device to receive and process an I/O request, acquire the requested data and prepare to start transmitting the data. The data transfer rate for an I/O device is defined as the number of bytes per second that it can transmit once the requested data has been acquired.
The read access delay and data transfer rate for two different I/O systems A and B are given the table below:

I/O System	Read access delay	Data transfer rate
A	5 seconds	5120 bytes/sec
B	3 seconds	3072 bytes/sec

Both systems employ an 8-bit bus to transfer I/O data. Each system is used to transfer a series of data blocks each of which contains 15360 data bytes. The read access and data transfer for each block occur sequentially. What is the minimum time between consecutive interrupts on:

a) (3) System A that employs a DMA controller? Minimum time = ___8 seconds___
One interrupt is generated at the conclusion of each block with DMA. The time interval between consecutive blocks on system A is 5 + 15360/5120 = 8 seconds.

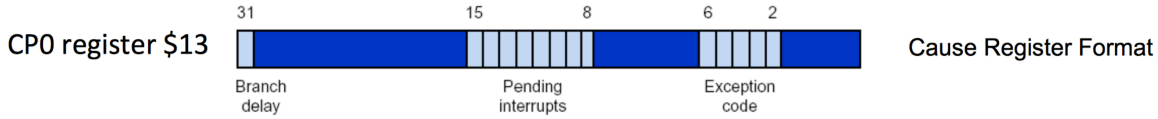
b) (3) System B which, once the data has been acquired, employs interrupt driven I/O using a single I/O port to transfer the data one byte at a time?
Minimum time = _325.52 micro-seconds __
System B takes 3 seconds to acquire the data, which it then transfers one byte at a time with an interrupt after each byte. System B transfers 3072 bytes per second. Therefore the minimum time between consecutive interrupts = 1/3072 = 325.52 micro-seconds.

2. A certain processor consumes 1000 cycles to perform the context switch required to transfer control to the interrupt handler for an I/O device that triggers an interrupt. The interrupt handler takes an additional 10,000 cycles to service the device request. Once the device has been serviced, another 1000 cycles are required to perform the context switch needed to return from the interrupt handler back to the program that was running. The processor's clock rate is 3 GHz.

a) (5) What is the maximum number of requests per second that a device can generate if the system must complete all activity associated with one interrupt before the next interrupt occurs?
Each interrupt requires 1000 + 10000 + 1000 = 12000 cycles. So no more than one request every 12000 cycles can be supported. Hence the upper limit is one request every 12000/3*10^9 = 4 micro-seconds which is 250000 requests per second.

(over)

b) (5) The registers used in managing exceptions and interrupts on our MIPS processor are described in the sub-module "2.2 Interrupt Driven Input/Output" as well as in section A.7 of the textbook.
Write down a short sequence of MIPS assembly language instructions to place just the current exception code right justified into CPU register $t0.
The exception code is contained within the cause register ($13) of coprocessor 0 (CP0).



The following instructions put the exception code into $t0:

```
mfc0   $t0,$13      ; copy cause register into $t0
sll    $t0,$t0,25   ; shift out bits on the left of the exception code field
srl    $t0,$t0,27   ; right justify the 5-bit exception code within $t0
```

3. (5) A disk rotates at a rate of 7200 revolutions per minute. Seek operations (i.e., moving the access head to a desired track) take on average 20 milli-seconds. Once the access head is on the appropriate track, it takes an average of one half revolution of the disk to get to the beginning of a requested sector. Each track contains 128 sectors and each sector contains 512 bytes of data. How long on average does it take to get the access head to the beginning of a randomly selected sector on a randomly selected track? Express your answer in micro-seconds (not seconds or milli-seconds).
The total average time required to get the access head to the beginning of a randomly selected sector = the average seek time + the average rotational latency
The time required for one revolution is 60/7200 seconds. On average the disk must make ½ rotation to get to a randomly selected sector. So the average time required to rotate to a random sector = ½ * (60/7200) = 4166.67 micro-seconds.
Therefore the total time = 20000 + 4166.67 = 24166.67 micro-seconds.

4. Assume that the 4 data strips (also called data blocks) within a certain stripe on a RAID 5 system are designated B0, B1, B2 and B3. The corresponding parity block P_{old} for this stripe is computed as the cumulative XOR of the 4 data strips within the stripe: P_{old} =B0 ^ B1 ^ B2 ^ B3. Hence each stripe contains a parity block and 4 data blocks. Block B2 is to be overwritten with the new contents designated as $B2_{new}$. None of the parity or data on the disk is currently known or in memory.

a) (5) Write down an expression for the corresponding new parity block P_{new} that requires reading no more than two blocks of any type from the 5 blocks in the stripe.
P_{new} = $B2_{old}$ ^ P_{old} ^ $B2_{new}$ ___ (where $B2_{old}$ and P_{old} are the two blocks read).

b) (5) Write down a different alternate expression for the new parity block P_{new} if there are no restrictions on the number or type of blocks read from the stripe.
P_{new} = __ B0 ^ B1^ $B2_{new}$ ^ B3_ (B0, B1, and B3 are read. B2new is already known)

5. (10) A disk system is to be constructed using some number of identical disk drives each of which holds 16 terabytes of data. A large database of size 96 terabytes is to be stored on the disk system. What is the minimum number of disks (data disks plus parity disks) that are required for the system if the disk system is a:

a) RAID6 system? 8 disks (96/16 = 6 for data plus 2 for parity)
b) RAID5 system? 7 (containing data and distributed parity)
c) RAID4 system? 7 (6 for data and one for parity)
d) RAID1 system? 12 (6 for the data and 6 for the mirror images)
e) RAID0 system? 6 (6 for the data and none for parity)

6. Sixty percent of the time required to complete a program on a certain computer system corresponds to the execution of code not related to I/O. The remaining 40% of the time required is due to I/O operations. The total time required to complete the entire workload is 480 seconds. This total time is to be reduced by 160 seconds by either improving CPU speed or by improving I/O.

a) (5) If the CPU is made N times as fast as the original CPU, what value of N yields the desired 160-second reduction in the total time consumed by the program? N= _2.25_. Express your answer to two decimal places (dddd.dd).

Reducing the total time by 160 seconds takes it down to 320 seconds.
The speedup S = 480.0/320.0 = 1.5
Let k be the improvement factor for the modified feature (CPU or the I/O system) and let f be the fraction of the workload performed by the modified feature. From Amdahl's law we know that the speedup S = T/ ((1-f) + f/k)T = 1/ ((1-f) + f/k)
So (1-f) + f/k = 1/S = 1/1.5
f/k = 1/S – (1-f)
k = f / (1/S – (1-f)) = 0.6 / (1/1.5 – 0.4) = 0.6 / (2/3 – 4/10) = 0.6 / (20/30 – 12/30)
= 0.6 / (8/30) = 18/8 = 2.25
So the CPU would have to be 2.25 times as fast.

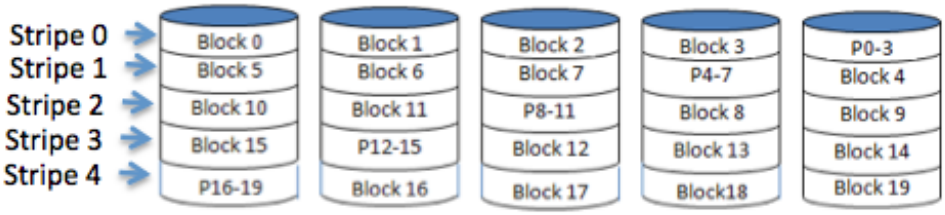
Alternatively, the time due to the execution of the code is 480*0.6 = 288
288/(288-160) = 288/128 = 2.25

b) (5) If instead of making the CPU faster, the I/O is made M times as fast as the original I/O, what value of M yields the desired 160-second reduction in the total time consumed by the program? M= _6_. Express your answer to two decimal places (dddd.dd).

Again k = f / (1/S – (1-f)) = 0.4 / (1/1.5 – 0.6) = 0.4 / (2/3 – 6/10)
= 0.4 / (20/30 – 18/30) = 0.4 / (2/30) = 12/2 = 6
So the I/O would have to be 6 times as fast.

Alternatively, the time due to I/O is 480*0.4 = 192
192/(192-160) = 192/32 = 6.00

7. Shown below is the diagram of a RAID system.

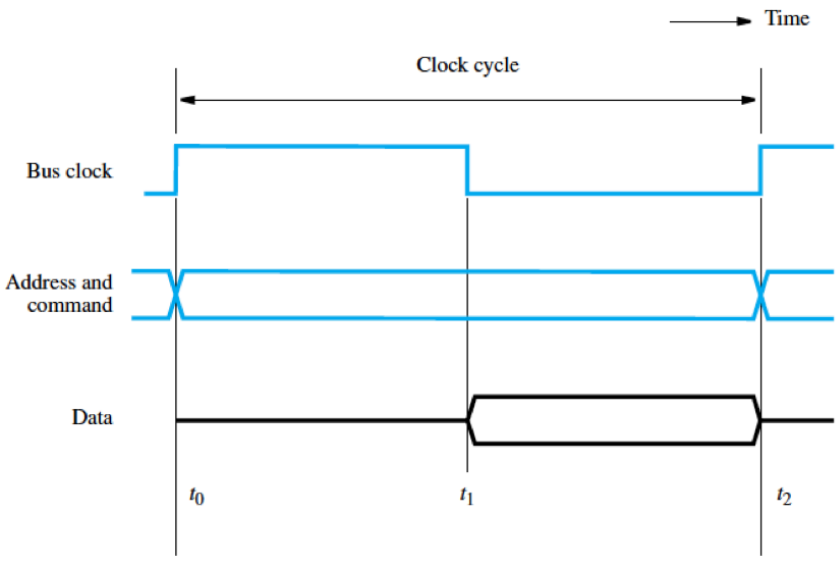


None of the information contained on the disks is currently in memory.

a) (5) What is the minimum number of disk blocks of any type that must be read to determine whether there is an error in some block within stripe 1?
All five blocks within the stripe would have to be read. The parity for the stripe can be computed based on blocks 4, 5, 6, and 7 and the result can be compared with the contents of the parity block P4-7 that was read from the stripe. A non-zero difference indicates that there is an error within the stripe.

b) (9) Block 2 within stripe 0 is to be updated and overwritten, while minimizing the number of blocks that are read and the number of blocks that are written. List the disk blocks of any type that must be read and those that must be written to accomplish this. For example: Block6, Block12, P15-16, etc.
Two blocks must be read: the old block2 and the parity block P0-3. The new parity is computed as old_block2 ^ old P0-3 ^ new block2. Two blocks must then be written: the new block2 and the new P0-3.

8. (10) Shown below is a copy of the simplified timing diagram from module 10 for a read operation over a synchronous bus.



Assume that zero wait states are required and that the bus clock has a 50% duty cycle; a 50% duty cycle means that the high and the low phases of the clock are of equal duration. The bus master takes 1.5 ns to place an address on the address lines. Once a slave device receives an address, it requires 3 ns to decode the address and a maximum of 5 ns to place the requested data on the data lines. The maximum propagation delay (the time required for information to pass from one device to another) on the data lines of the bus is 4 ns. Once the data arrives at the requesting device, it takes 1 ns to transfer the data from the bus into the device's data register. What is the maximum bus clock rate that can be used with this system?

The maximum bus clock rate would be the reciprocal of the minimum time interval from t_0 to t_2. The minimum time for the high phase of the clock is the time for the address to be placed on the address lines, arrive at the slave and be decoded (1.5 + 4 + 3 = 8.5 ns).

The minimum time for the low phase of the clock is the time for the slave to acquire and place the requested data on the data lines, for the data to arrive at the master and for the master to transfer the data into its device register. The total time required for phase two is 5 + 4 + 1 = 10 ns. Since the two phases must be equal (by definition of the 50% duty cycle), they both must be allocated the larger time (10 ns).
So the minimum value for the interval t_0 to t_2 = 2 * 10 ns = 20 ns.
Therefore the maximum bus clock rate is 1/20ns = 50 MHz.

9. (16) Shown below is a RAID-DP (i.e., RAID6 with dual parity) disk system. D1, D2, D3 and D4 are the data disks. RP is the horizontal or row parity disk and DP is the diagonal parity disk. For the purposes of this problem, each strip or block on a disk is a 4-bit pattern. Two of the disks, D2 and D4 have crashed so their data contents are unknown.
Module 10 example set 7 illustrates a technique to reconstruct blocks on up to 2 disks that have failed and are unavailable. Use the scheme described in the example set, substituting exclusive-OR in place of addition, to reconstruct the contents of the two missing disks by determining the 4-bit pattern for each of the following:

D2_blue =	_0100_	D4_red=	__1001_
D2_purple=	_1001__	D4_orange=	__0101_
D2_white=	_0011__	D4_blue=	__0111_
D2_red=	_0001__	D4_purple=	__1111_

List your answers in the order shown above.

D4_orange = 1010 ^ 1100 ^ 1101 ^ 1110 = 0101
D2_purple = 1011 ^ 0010 ^ 0101 ^ 0101 = 1001
D4_purple = 1101 ^ 1001 ^ 0101 ^ 1110 = 1111
D2_red = 1010 ^ 0110 ^ 1111 ^ 0010 = 0001
D4_red = 1100 ^ 0001 ^ 0101 ^ 0001 = 1001
D2_blue = 1101 ^ 1100 ^ 1001 ^ 1100 = 0100
D4_blue = 0100 ^ 0010 ^ 0010 ^ 0011 = 0111
D2_white = 1100 ^ 0101 ^ 0111 ^ 1101 = 0011

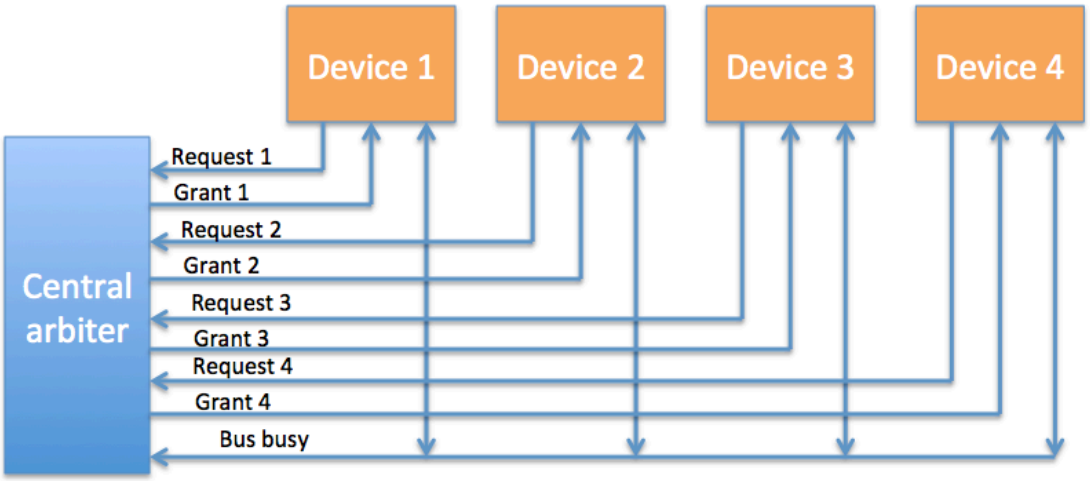
10. Shown below is a bus system with a central arbiter that handles four devices. The device priorities range from highest for Device 4 down to the lowest for Device 1. The arbiter grants access to the highest priority request that is present. Each device has a separate request and grant line. The total available bus bandwidth that must be shared by the devices is 80 MB/s. Each device wants to use 30MB/s of bus bandwidth. All of the devices continuously compete for use of the bus and transmit blocks of the same size for each transaction. If a device is not granted access to the bus, it tries again as soon as the bus is not busy and keeps trying until it gets access. How much bus bandwidth is Device 1 able to use and how much bus bandwidth is Device 3 be able to use if:

a) (5) Bus access is granted strictly on a priority basis?
Device 1 bandwidth = ____0___ Device 3 bandwidth = ___30MB/s_____
Device 4 has the highest priority and gets all of the 30 MB/s bandwidth that it requests. This leaves 80 – 30 = 50 MB/s for the others. Device 3 will get its 30 MB/s leaving only 20 MB/s for Device 2. Device 1 gets no bandwidth since none is leftover.

b) (4) Instead of strict priority, the bus arbiter uses a fairness policy to give each device an equal chance to use the bus?

Device 1 bandwidth = __20 MB/s__ Device 3 bandwidth = __20 MB/s__

With the fair bus arbitration policy, any device that is not given access will have higher priority than the device it lost to until it gets a chance to use the bus. Access to the bus will be granted on a round robin basis. Hence, the 80 MB/s of available bandwidth will be evenly divided between the four devices and each will get (80 / 4) = 20 MB/s of bus bandwidth.



Computer Science 605.611

Problem Set 12 Answers

1. A certain program consists of three independent parts A, B and C that can execute in parallel on separate processors within a multiprocessor system. None of the parts can be further subdivided and each part must execute on a single processor. Part A requires 78 billion cycles to complete, part B requires 40 billion cycles to complete and part C requires 62 billion cycles. The program is executed on an SMP system with 4 identical processors all running at the same 2 GHz clock rate.

a) (5) How long does it take to execute the entire program (all 3 parts) if only one of the processors is used to execute the program (the other 3 processors remain idle)?
The single processor consumes (78+40+62) billion cycles = 180 billion cycles. This corresponds to 180 billion cycles/(2 billion cycles/sec) = 90 seconds.

b) (5) What is the minimum time required to execute the entire program (all 3 parts) if only two of the processors are used to execute the program (the other 2 processors remain idle)?
One processor executes part A in 78 billion cycles while the other processor executes part B. Part B is completed after 40 billion cycles. The second processor could then execute part C. Hence the minimum time to execute the program corresponds to Max(78, (40+62)) = 102 billion cycles. So it takes 102/2 = 51 seconds. Doubling the number of processors does not cut the execution time in half.

c) (5) What is the minimum time required to execute the entire program (all 3 parts) if all 4 processors are used?
In this case, parts A, B and C could each be executed in parallel by three separate processors. Therefore the minimum time to complete the program would correspond to the longest part (78 billion cycles). The required execution time = 78/2 = 39 seconds.

2. Another different program consists of a purely sequential part that requires 20 seconds to execute on a single-core processor. The sequential part must execute first and is followed by a parallel part containing 4 independent indivisible tasks each of which takes 12 seconds and can only run on a single processor.

a) (5) What is the total execution time for the entire program on the single-core processor?
Execution time on single-core = 20 + 4*12 = 68 seconds.

b) (5) Compared to the single-core processor, what is the actual speedup achieved by running the program on a 4-core version of the processor?
Execution time on 4-core system = 20 + 12 = 32 seconds
Speedup = 68/32 = 2.125

c) (5) What speedup is predicted by Amdahl's law for the program running on the 4-core system compared to the single-core system?
Recall that for multi-processor or multi-core systems Amdahl's law states:

Speedup = N / [f + N(1-f)]

where f is the fraction of the total execution time accounted for by the parallel part and N is the number of cores on the multi-core system.

From part a) the total execution time for the program on a single core is 68 seconds. The sequential part requires 20 seconds and the parallel part accounts for the remaining 48 seconds.
Hence f = 48/68 and N=4
Speedup = 4/[48/68 + 4*20/68] = 4/[128/68] = 4*68/128 = 272/128 = 2.125
Which is the same speedup computed in part b).

3. (10) A different program is to compute two products: one is a vector product of two vectors each containing 100 elements; the other product is produced by multiplying all of the elements of a matrix by a single scalar constant. The matrix has 1000 rows and 1000 columns. The vector product must be computed first and each of its elements is the product of the corresponding elements in the two vectors. That is, P[i] = A[i]*B[i] where P is the product vector and A and B are the two vectors that are multiplied. Multiplying one vector element by another takes 1 cycle and multiplying the scalar by a single matrix element takes one cycle. The program is run first on a single processor. What speedup is provided for the same program if 499 additional identical processors are included so that the system has a total of 500 processors and there are no memory conflicts or other dependencies?

The vector product requires multiplying the corresponding elements from the two vectors, this takes 100 cycles on a single processor. The matrix contains 1000*1000 = 1000000 elements. So, multiplying the matrix by a scalar takes 1000000 cycles. Hence a single processor takes 1000100 cycles.

With 500 processors, the vector product could be computed in just one cycle by using a different processor for each element in the vector product (the other 400 processors would be idle). For the matrix product, 500 processors could multiply the scalar by one half of the elements in each of the rows of the matrix at the same time.
Hence multiplying the scalar times all 1000 elements in each row of the matrix takes 2 cycles. So multiplying the scalar times all 1000 rows takes 1000*2 = 2000 cycles. Therefore the sum of the times for the vector and matrix products is 2001 cycles.
This yields a speedup = 1000100/2001 = 499.8

4. A dual processor SMP system includes an L1 data cache for each processor and employs the MESI protocol to maintain cache consistency. Each cache is a 2-way set associative copy-back cache that contains a total of 8192 cache lines each of which is 256 bytes in size. The lines or ways within each empty set are filled in the order Way0, Way1, Way2 then Way3. A write-allocate policy is used for each cache. One process, P1, runs on the first processor at the same time that another process, P2, runs on the other processor. The first access made is when P1 reads a variable X that resides in memory at address 0x400804C0 and contains an initial value of 80. After P1 reads X, P2 writes the value 156 into a variable Y located at memory address 0x400804F8. All caches are initially empty, so each cache line starts out in the invalid (I) state.

a) (5) After P1 first reads X, what is the MESI state of the line containing X in P1's cache and how is P2's cache affected? That is, what change (if any) occurs in the MESI state for the affected line or lines in the two caches?

Each set contains 2 lines. There are 8192/2 = 4096 sets in the cache. The variable X resides within memory block 0x400804 which maps to set 0x0804 = 2052 in P1's cache.
Since this is the first read from this block, the state of the line in P1's cache changes from invalid (I) to exclusive (E) and there is no change to P2's cache.

b) (5) After P1 first reads X and P2 then writes Y, what is the MESI state of the line containing Y in P2's cache and how is P1's cache affected? That is, what change (if any) occurs in the MESI state for the affected line or lines in the two caches?

The variable Y resides within memory block 0x400804 as well and maps to set 2052 in P2's cache. The state of the line containing a copy of this memory block in P2's cache changes from I to M. The line containing a copy of the same block in P1's cache changes from E to I.

5. (15) A program runs on a superscalar uniprocessor with a 2 GHz clock rate within a NUMA (non-uniform memory access) system. The number of instructions executed in the program is IC. Each instruction that does not reference remote memory takes 2 cycles. Each instruction that does reference remote memory incurs an additional 200 ns penalty over those that do not reference remote memory. The percentage of program instructions that access remote memory is 0.2%. What speedup is achieved for the program if the remote memory access penalty is somehow reduced from 200 ns to 10 ns per access?

The 2 GHz clock rate corresponds to a 0.5ns cycle time, so the extra 200 ns corresponds to 400 clock cycles. Instructions that do not reference remote memory take 2*0.5ns = 1ns to complete.
Speedup = T1/T2 where T1 is the execution time with the 200ns remote memory access penalty and T2 is the execution time with the 10 ns remote memory access penalty.
T1 = IC*1ns + IC*0.002*200ns = IC*(1.0ns + 0.4) = IC*1.4ns
T2 = IC*1ns + IC*0.002*10ns = IC*1.02ns
Therefore the speedup = 1.4/1.02 = 1.3725

6. (15) A uniprocessor system accesses memory over a 32-bit bus. The processor has a direct mapped write-back (as opposed to write-through) data cache that operates in look-through mode and employs a write-allocate policy. The data cache contains 65536 lines, each of which is 1024 bytes in size. Detecting a cache miss, or performing a cache read, or performing a cache write for the data cache each takes 2 CPU clock cycles. Loading a memory block into the data cache or writing a data cache line back to memory takes 400 CPU clock cycles. Recall that for a cache miss, the data item that is needed from the memory block is transferred to the CPU in parallel with loading the memory block containing the data item into cache.

Each of the 134217728 four-byte elements in an integer array are read and updated by the code shown below. The address of the array in memory is 0x10084000.

```
        lui   $8,0x1008      #point to the first element to be processed
        ori   $8,0x4000
        lui   $4,0x0800      # number of elements = 134217728 (=0x08000000)
loop:   lw    $12,0($8)      # get next element
        sub   $12,$0,$12     # negate by subtracting from 0
        addi  $8,$8,4        # point to next element
        sw    $12,-4($8)     # update the element that was read
        bgtz  $4,loop        # repeat if more elements remain
        addi  $4,$4,-1       # decrement loop control variable
```

In an attempt to speedup the processing of the array, the code is executed on an 8-core system in which each core has a separate data cache identical to the data cache for the uniprocessor. A speedup factor of 8 for the 8-core system compared to the uniprocessor is defined as "linear" speedup.

What is the actual speedup provided by the 8-core system based on just the time required to read and update all of the array elements? Ignore the time required by the instructions that do not reference memory as well as any possible memory conflicts and the flushing of the caches once the code completes. Each of the 8 cores processes a separate contiguous subset of elements by executing a separate copy of the code sequence. Each subset corresponds to $1/8^{th}$ of the array. Assume that all data caches are initially empty and that the final cache flush when the program ends is handled by the operating system (so it does not contribute to the code's execution time).

Reading an array element that is not already in the cache requires 2 cycles to detect the miss plus 400 cycles to load the memory block containing the element into cache while returning the element to the CPU.
Each write of an array element is a hit, since elements are only written after they have been read. So 2 cycles are required to write the element into cache. Updating the first element in each line

requires 2 + 400 + 2 = 404 cycles. Updating each element that is already in the cache requires 4 cycles (2 cycles for the read hit plus 2 cycles to write the updated element).

So updating all elements in a line requires 404 + (N-1)*4 cycles where N is the number of elements in the line. That is, 404+255*4 = 1424 cycles per line.

Each block contains 1024/4 = 256 elements. The starting address 0x10084000 corresponds to the beginning address of a memory block. The total array requires 134217728/256 = 524288 memory blocks. Updating the first 65536 blocks fills the cache for the first time and takes a total of 65536*1424 = 93323264 cycles.

Thereafter, each block loaded into the cache must replace a line already in the cache.
Each replacement adds 400 cycles to the total time.
Hence the next 524288-65536 = 458752 blocks requires 400+1424 = 1824 cycles to update the array elements contained in the block.
The grand total number of cycles required to update the entire array on the single core system = 93323264 + 458752*1824 = 930086912 cycles

With the 8-core system, each core updates 1/8 of the array elements (=524288/8 or 65536 elements). So each core only has to fill its cache once and none of the lines have to be replaced. It takes 65536 * 1424 = 93323264 cycles to update the 8 subsets of array elements in parallel.

This corresponds to a speedup of 930086912 /93323264 = 9.97

This is greater than the linear speedup of 8 and is referred to as “superlinear” speedup.

7. (5) Use our previously described 3-bit pseudo-LRU replacement algorithm for a 4-way set associative cache as a guide for designing a pseudo-LRU replacement algorithm for an 8-way set associative cache. What is the minimum total number of pseudo-LRU bits required for each set in the 8-way set associative cache? Describe how the pseudo-LRU bits are used with the 8-way set associative cache. You do not need to specify the actual bit patterns.

Seven bits per set are needed; one bit is used to divide the 8-way system into two 4-way subsystems, and a separate 3-bit group is used for each of the two 4-way subsystems. The minimum number of pseudo-LRU bits required per set = 1 +2*3 = 7.

8. Answer the following questions about cache coherency protocols in a dual-processor system.

a) (5) If each processor has a separate cache, what is the main advantage of the 4-state MESI snoopy cache coherency protocol compared to a snoopy cache coherency protocol that uses only 3 states M, S and I (modified, shared and invalid)?

With the MESI protocol, the first time that a memory block is loaded into any cache its state is changed from invalid to exclusive. The exclusive state indicates that the newly loaded line resides in no other caches so a write to the line does not have to be broadcast to other caches. This reduces the traffic on the shared bus. Without the exclusive state, the state of the newly loaded line would change from invalid to shared. So the shared state would now indicate that the line is in the local cache and may be in one or more other remote caches. A write to the cache line would have to be broadcast on the bus to alert the other caches just in case they contain a copy of the line.

b) (5) Suppose that processor P1 (in a dual-processor system) writes to a memory block. A copy of this same memory block is already in a modified line within the local cache of processor P2. What actions should be taken for the modified line within P2's cache? Explain your answer.

Even though the line has already been written to locally in P2's cache, the write from a remote processor (P1) could be to a different location within the line than the location that was written to by P2. Hence a back-off signal should be sent to the remote processor (P1) and P2's modified local cache line is written back to memory and marked as invalid. The remote processor (P1) can then complete its write operation. So P2's cache line will be invalid after P1 completes its write.

c) (5) Assume that the 3-state MSI cache coherency protocol is used with a single 2-way set associative cache containing a total of 1048576 lines each of which is 512 bytes in size. What is the minimum total number of MSI state bits needed for the entire cache?

The state must be recorded for each cache line. Since there are three states (M,S, I), at least 2 bits are needed to encode the state. Therefore the total number of MSI bits required for the cache is 2*2097152= 4194304.

Computer Science 605.611

Problem Set 13 Answers

1. a) (5) Show a pair of Intel IA-32 instructions that place the 64-bit two's complement equivalent of the integer -763 (negative 763) into registers EAX and EDX (with the high part in EDX).

```
MOV   EAX,-763      put -763 into EAX
CDQ                 convert 32-bit value in EAX into 64-bits in EDX,EAX
```

b) (5) Show a pair of SparcV8 instructions that place the 64-bit two's complement equivalent of the integer -763 (negative 763) into registers %o4 and %o5 (with the high part in %o5).

```
add    %g0,-763,%o4
add    %g0,-1,%o5
```

2. A function or procedure that is called usually performs some task and returns control to the caller. Indicate where (i.e., in which register of memory location) the return address is saved by each of the following processor instructions:

a) (5) SparcV8 instruction: call sqrt

Register %o7 (linkage register) is used for the return address = %o7+8

b) (5) Intel IA-32 instruction: call sqrt

The return address is pushed onto the stack.

c) (5) ARM instruction: BL sqrt

The return address is placed into R14, the linkage register.

3. a) (4) Show a MIPS true-op instruction that implements the nop synthetic or pseudo-instruction.

```
sll    $0,$0,0
```

b) (4) Show a SparcV8 true-op instruction that implements the nop synthetic or pseudo-instruction.

```
sethi  %hi(0),%g0
```

4. (5) The IEEE-754 single precision and double precision floating point formats provide 7 and 15 decimal digits of accuracy, respectively. How many decimal places are provided by the Intel internal 80-bit extended precision floating point format?

Show how you arrived at your answer.

One bit is used for the sign and 15 bits are used for the exponent. One bit is used to indicate whether the number is normalized or denormalized. This leaves 63 bits for the value which corresponds to $63*\log_{10}(2) = 63*0.30103 = 18.965$ (i.e., about 19 decimal places).

5. (5) Manually generate (i.e., by hand) the 80-bit extended precision floating point representation of the negative value: -4.87493 on an IA-32 processor. Show your answer as a sequence of hex digits.

The integer part in binary is 100.

0. 87493 * 2^{64} = 16139609792410697728 = 0xDFFB69984A0E4000

So decimal -4.87493 = - 0x4.DFFB69984A0E4000 = negative binary

$1.0011011111111110110110100110011000010010100000111001000000000000000 * 2^2$

The excess-16383 representation of the exponent is 16383+2 = 16385 = binary 100000000000001.

The 80-bit pattern is:

1100 0000 0000 0001 1001 1011 1111 1111 0110 1101 0011 0011 0000 1001 0100 0001 1100 1000 0000 000

= 0xC0019BFF6D330941C800

6. (5) The following MIPS instruction sequence saves MIPS registers $2, $4 and $6 on the stack:

```
add   $sp,$sp,-12
sw    $2,8($sp)
sw    $4,4($sp)
sw    $6,0($sp)
```

Show a single (i.e., one) complete ARM assembly language instruction that similarly saves ARM registers R2, R4 and R6 on the stack.

STMFD R13! , {R2, R4, R6}

7. a) (5) Show one MIPS true-op instruction that has the same effect as the instruction: LEA EAX,[EBX + 44] on the Intel IA-32 processor. Use $8 as the result register and $9 as the base register in the MIPS instruction.

addiu $8,$9,44

b) (5)) Show one ARM true-op instruction that has the same effect as the instruction: LEA EAX,[EBX + 44] on the Intel IA-32 processor. Use R4 as the result register and R5 as the base register in the ARM instruction.

ADD R4,R5,#44

8. (5) The following ARM processor instruction: MVN R2,#3 places a bit pattern into register R2. If the resulting bit pattern is interpreted as a two's complement format signed integer, what signed decimal (i.e., base 10) value does it represent?

Contents of R2 represents: – 4_____ Bit pattern = __0xFFFFFFFC _

9. The acronym IBM stands for International Business Machines. What does each of the following acronyms stand for? Each acronym relates to one of the processors described in this course.

a) (1) MIPS - Microprocessor without Interlocking Pipeline Stages

b) (1) ARM - Advanced RISC Machine

c) (1) IA-32 - Intel Architecture 32

d) (1) Sparc - Scalable Process Architecture

10. Assume that the MMX registers MM2 and MM4 on an IA-32 processor contain the 64-bit patterns 0xA0A1A2A3A4A5A6A7 and 0xB0B1B2B3B4B5B6B7, respectively. Assume also that the MIPS registers $2, $3, $4 and $5 contain the following bit patterns:

$2	$3	$4	$5
0xA0A1A2A3	0xA4A5A6A7	0xB0B1B2B3	0xB4B5B6B7

a) (5) Show, in hex, the contents of register MM2 after the IA-32 instruction
PADDB MM2,MM4 is executed. Result in hex = 0x50525456585A5C5E.

b) (5) Show, in hex, the contents of MIPS registers $2 and $3 after the following two MIPS instructions are executed:
addu $2,$2,$4
addu $3,$3,$5
$2 in hex = 0x51535556___ $3 in hex = __0x595B5D5E___.

11. A machine instruction stores the 32-bit contents of a register into memory at address 0x80084004. Show the contents of the single byte at location 0x80084006 if the store instruction is executed on:

a) (5) an IA-32 processor and the register that is stored contains 0xFEEDBEEF ?
Contents of byte in hex = 0xED
The IA-32 system employs little-endian memory storage order, so byte 0 from the register (the low byte) which contains 0xEF is stored at 0x80084004 , and byte 2 which contains 0xED is stored at location 0x80084006

b) (5) a SparcV8 processor and the register that is stored contains 0xA1C0FFEE ?
Contents of byte in hex = 0xFF
The Sparc employs big-endian memory storage order, so the high byte from the register (which contains 0xA1) is stored at 0x80084004 and the byte that contains 0xFF is stored at location 0x80084006.

12. a) As described in module 13, what is the maximum number of CPU registers contained in each of the following processors?

- (2) IA-32 8 operand registers: EAX, ECX, EBX, EDX, EBP, ESP, EDI and ESI

  (2) ARM 16 operand registers: R0 – R15 used in the User/System mode. Additional "banked" registers are available in each of the 5 exception modes. The additional registers are outlined in blue in the diagram below. These replace some of the corresponding user/system registers depending on the exception mode: 7 for FIQ and 2 for each of the other four modes. Hence the total number of registers that can by used as operands by the ARM CPU = 16 + 7 + 2 + 2 + 2 +2 = 31.

Accessible registers in different modes of the ARM processor.

User/System	FIQ	IRQ	Supervisor	Abort	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_irq	R13_svc	R13_abt	R13_und
R14	R14_fiq	R14_irq	R14_svc	R14_abt	R14_und
R15	R15	R15	R15	R15	R15

- (4) Sparc_v8 There are 8 global registers (%g0 - %g7). The first window contains 8 in registers (%i0 - %i7), 8 local registers (%l0 - %l7), and 8 out registers (%o0 - %o7). For each window after the first, the in registers (%i0 - %i7) overlap with and are the same as the out registers (%o0 - %o7) of the previous window. Hence each window after the first requires 16 new registers. Windows 2 through 31 contain 24 registers (8 are from the previous window and 16 are new).
  So the total (for 32 register windows) is 8 + 8 + 8+ 8 + 31*16 = 528.

b) The PC (program counter) is to be incremented by 16 using a single ADD instruction. For each of the processors listed below, indicate whether the processor can or cannot accomplish this.

- (2) ARM processor Yes. R15 is used as the program counter, so it can be incremented using an ADD instruction.
- (2) MIPS processor No. The PC can't be used as an operand on the MIPS. So an ADD instruction can't be used to increment it by 16.
- (2) IA-32 processor No. The IA-32 EIP register is the program counter and can't be used as an operand for an ADD instruction.
- (2) Sparc V8 processor No, the Sparc processor can't use the PC as an ALU operand.