ECE 2035 Programming HW/SW Systems Fall 2016

4 problems, 6 pages Exam One Solutions 21 September 2016

Problem 1 (2 parts, 30 points) Loops

Part A (12 points) Suppose the following declaration is given of an array `Timeline[10]` of alternating year, city integers, as in Homework 2. Assume the initial value shown below is just an example; it can be initialized with a different set of years/cities, under the following constraints. The years range from 1986 to 2015, inclusive, and each year indicates when a move was made to a new city. The total number of moves in each timeline is exactly five. Write a program that computes the total number of moves (`NumMoves`) that were made in the 1990's. ***For maximum credit, declare and initialize any necessary variables.***

```
int  Timeline[10]  =  {1987,  2,  1993,  9,  ...  2014,  7};  //  given
int NumMoves=0;
int i;
for (i = 0; i<10; i += 2)
   {
      if ((Timeline[i] > 1989) && (Timeline[i] < 2000))
         {
            NumMoves++;
         }
   }
```

Part B (18 points) Write MIPS code for the fragment in Part A. **Store the `NumMoves` computed in register $2**. *For maximum credit use a minimum number of instructions.*

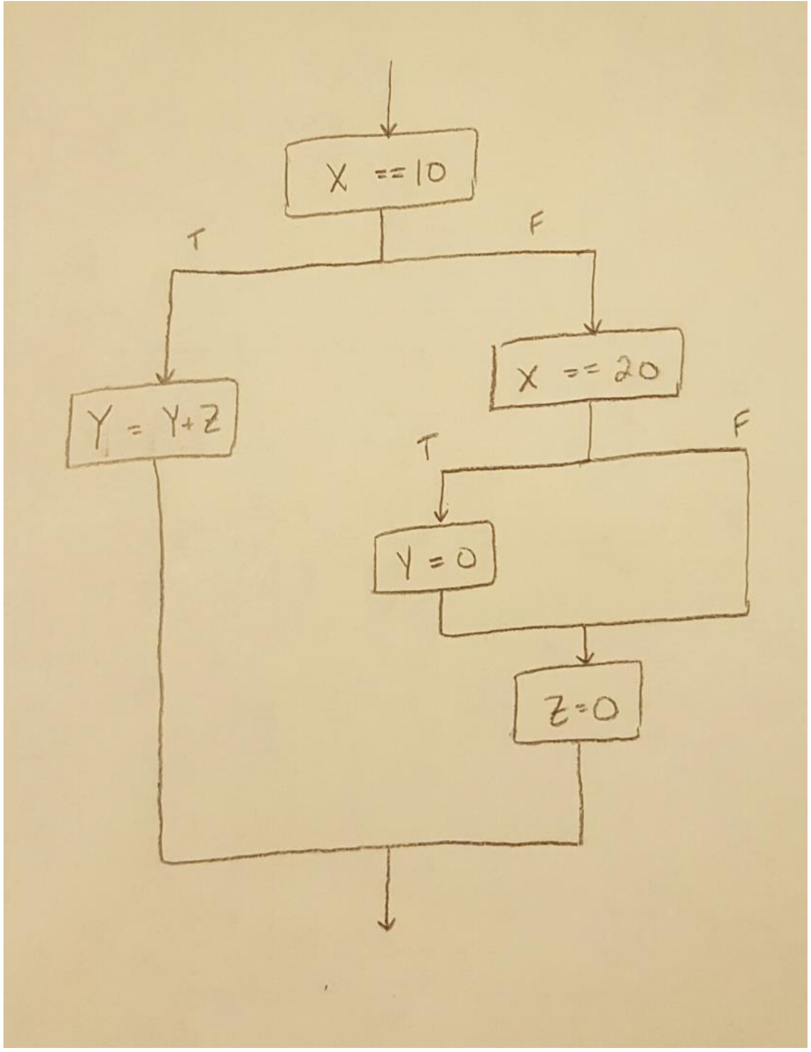
Label	Instruction	Comment
	.data	#
TimeLine:	.word 1986, 4, 1988, …, 2	# given Timeline
	.text	#
	addi $1, $0, 0	**# init i**
	addi $2, $0, 0	**# init NumMoves**
Loop:	**slti $3, $1, 40**	**# is i<10?**
	beq $3, $0, Exit	**# if not, Exit loop**
	lw $4, Timeline($1)	**# read Timeline[i], a year**
	slti $3, $4, 1990	**# is year < 1990?**
	bne $3, $0, Update	**# if so, update i, keep looping**
	slti $3, $4, 2000	**# is year < 2000?**
	beq $3, $0, Update	**# if not, update i, keep looping**
	addi $2, $2, 1	**# else, increment NumMoves**
Update:	**addi $1, $1, 8**	**# update offset into Timeline**
	j Loop	**# loop back**
Exit:	**jr $31**	**# return**

Problem 2 (2 parts, 24 points) **Conditionals: Nested if-then-else**

We have learned that there are many conditional structures, such as if-then-else, compound predicate assignments, ternary decision statements, and switch statements. For the following MIPS code, assume that $2, $3, and $4 are assigned to integers x, y and z respectively.

```
start:      addi  $1, $0, 10
            beq   $1, $2, L10
            addi  $1, $0, 20
            beq   $1, $2, L20
            j     Ld
L10:        add   $3, $3, $4
            j     end
L20:        addi  $3, $0, 0
Ld:         addi  $4, $0, 0
end
```

Part A (8 points) Draw the control flow graph for the MIPS code shown.



Part B (16 points) Write the C code that corresponds to the above MIPS code with (possibly nested) if-then-else statements, and **without** any if-then-else statements (hint: switch will work).

With if-then-else statements	**Without if-then-else statements**

`if (x ==10)`
`    y = y+z;`
`else`
`  {`
`    if (x == 20)`
`        y = 0;`
`    z = 0;`
`  }`
	`switch (x)`
	`  {`
	`    case 10:`
	`      { y = y+z;`
	`        break;`
	`      }`
	`    case 20:`
	`      y = 0;`
	`    default:`
	`      z = 0;`
	`  }`

`// another answer:`
`if (x ==10)`
`  y = y+z;`
`else if (x == 20)`
`         { y = 0;`
`           z = 0;`
`         }`
`else`
`  z = 0;`	

Problem 3 (3 parts, 21 points) **Assembly Programming**

Part A (8 points) The following buggy code fragment was supposed to sum up all the (unsigned) hexadecimal digits in the word stored at XLoc and to put the sum in register $5. Sometimes it gives the correct answer and sometimes it doesn't. Give an example initial value (in hex) stored at XLoc that will result in the correct sum and give an example initial value (in hex) where the resulting sum is incorrect.

Example initial value that gives **correct** sum:	**0x0*0*0*0* (where * is any hex digit)**
Example initial value that gives **incorrect** sum:	**0xABCD1234**

Label	Instruction	Comment
	.data	#
XLoc:	.word 0x__________________	#
	.text	#
	addi $7, $0, 0	# $7: byte offset
	addi $5, $0, 0	# $5: running sum
	addi $4, $0, 4	# $4: constant 4
Loop:	beq $7, $4, ExitLoop	# if $7 == 4, exit the loop
	lbu $3, XLoc($7)	#
	add $5, $5, $3	#
	addi $7, $7, 1	# increment byte offset $7
	j Loop	# loop back
ExitLoop:	...	# stuff after the loop

Part B (8 points) Suppose register $3 has 2 hexadecimal digits in its lower 8 bits (least significant byte) and zeros in the upper 24 bits (upper 3 bytes). Write a MIPS code fragment to sum the two hexadecimal digits and put the result in $5.

Label	Instruction	Comment
	andi $6, $3, 0xF	**# mask to extract lower 4 bits**
	srl $5, $3, 4	**# shift out lower 4 bits**
	add $5, $5, $6	**# add remaining hex digit to $6**

Part C (5 points)Write a **single** MIPS instruction that is equivalent to the original fragment. Assume *little endian* byte ordering.

Original:	Equivalent MIPS statement:
lui $4, 0xFF	**lbu $3, 1002($0)**
lw $3, 1000($0)	
and $3, $3, $4	
srl $3, $3, 16	

Problem 4 (2 parts, 25 points) **Nonlocal Control Flow**

Part A (12 points) What does the following code fragment print?

```
int i, x;
int A[10] = {99, 33, 44, 22, 56, 78, 1, 5, 9, 88};

for(i=0; i<10; i++){
  if(i & 3)               // bitwise AND
     continue;
  x = A[i];
  printf("x = %d\n", x);
}
```

```
x = 99
x = 56
x = 9
```

Fill in the blanks to rewrite the code above to produce the equivalent behavior without using `continue`.

```
int i;
int A[] = {99, 33, 44, 22, 56, 78, 1, 5, 9, 88};

for(i=0; i<9; i += 4){
  x = A[i];
  printf("x = %d\n", x);
}
```

Part B (13 points) Answer the three questions below about the following C fragment.

```
int i, j, k, count;
j = k = count = 0;

for (i=0; i < 8; i++){                      // outer loop
   if (i % 2) continue;
   // j = 0;            reinitialize j
   while (j < 8){                           // middle loop
          // k = 0;     reinitialize k
          while (k < 8){                    // inner loop
              if (k % 2) break;
               k++;
          }
          count++;
          j++;
   }
}
printf("%d\n", count);
```

How many times is `break` executed?	**8**	**32** *(assuming j,k reinitialized)*
How many times is `continue` executed?	**4**	**4**
What is printed?	**8**	**32** *(assuming j,k reinitialized)*