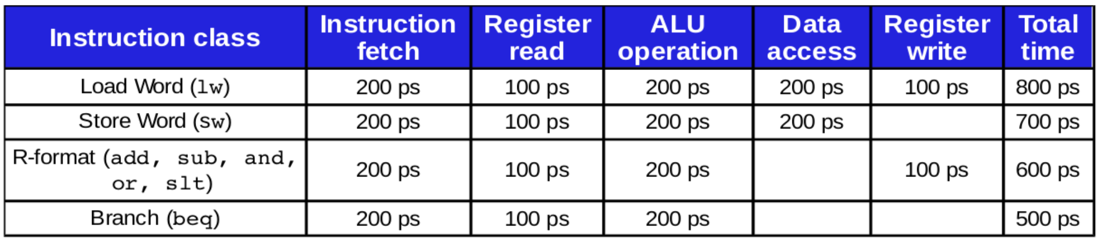
Computer Architecture and Programming Languages
Jacobs University Bremen
Jovan Shandro

Course: CO20-320241
December 2, 2019

Homework 11

Problem 11.1

Solution:
The data from slide 29 that is going to be used is the following table:



Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load Word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store Word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add, sub, and, or, slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

And in our example we have a program with 2 load instructions, 1 store instruction, 3 R-format instructions, and 1 branching instruction (7 in total).

a) We first start analysing the single cycle approach. In the single cycle approach, each of the instructions will be executed within a single clock cycle, and the clock cycle is equal to the time taken by the slowest instruction, which in our case as we can see from the table is the load instruction with a total time of 800ps. So this is the time it will take to execute every single instruction, and since we have 7 instructions in total, the overall time the program would take is $5600ps$.

Now we consider the multi-cycle approach. In the multi-cycle approach each of the instruction may be executed in more than one cycle. In this approach, for each of the instructions the clock time will be determined by the number of steps they are executed (found in the table) and the time taken for each will be determined by the slowest step, and as we see from the table the slowest step takes 200ps. The load instructions will need 5 cycles, the store instruction will need 4, all R-format instructions will need 4, and the branching instruction will need 3. So the total time taken for the program to execute will be: $200 \cdot (5 \cdot 2 + 4 \cdot 1 + 4 \cdot 3 + 3 \cdot 1) = 200 \cdot 29 = 5800ps$. So we notice that for this program the single cycle approach is faster by $200ps$.

b) We know from a) that the single cycle approach would take $5600ps$. Now we just need to analyse the pipeline approach.
In the pipeline approach (assuming there are no hazards), the instructions are also divided into 5 steps and the clock cycle takes once again $200ps$ as the slowest out of all the steps. In this approach the instructions run one after the other, so to find the total time, we multiply the max step time (200) by the number of instructions, plus the number of maximum steps for the instructions minus one (as a step is calculated twice), so we get the total time of $200 \cdot (7 + 5 - 1) = 200 \cdot 11 = 2200ps$. So the pipeline approach is faster than the single-cycle approach by $2400ps$.

c) We know from points a) and b) that the multi-cycle and pipeline approach take $5800px$ and $2200ps$ respectively, so for our program the pipeline approach is faster than the multi-cycle approach by $2600ps$.

Problem 11.2

Solution:
a) Since we just want to recognize the string `$zero`, the expression is: **`"$zero"`**

b) We want to recognize strings that start with 'a' and end with 'b', so the regular expression must start with 'a' and end with 'b'. To represent any character that can be a lowercase or uppercase letter or a digit, the expression would be [a-zA-Z0-9], and since the number of such characters in between 'a' and 'b' can be 0 or more, we put * in its end and we get the final regular expression: **a[a-zA-Z0-9]*b**

c) Since the pattern should start and end with a digit, the start and end of the regular expression will be [0-9]. Almost same as above, to match a character that could be a letter, a digit or an underscore the expression would be [a-zA-Z0-9_], and since there can be any number of such characters including 0 we use * in its end and thus the regular expression is:
[0-9][a-zA-Z0-9_]*[0-9]

d) Since the string should always start with "abb", "abb" will be the beginning of the regular expression. Then, since there must be at least 4 'a'-s and the length must be at most 10, the abb in the beginning must be followed by a{4,4} to make sure at least 4 a-s are there and then [ab]{0,3} to make sure there are at most 10 characters and that the string contains only 'a' and 'b' as characters. So the final expression is: **abba{4,4}[ab]{0,3}**

e) In order to form a positive integer we just need to put an arbitrary number of digits (we need at least one) and also take care of the fact that the first digit has to be non-zero, so the regular expression is: **[1-9][0-9]+**

f) To be able to match all integers, we also need to include negative integers in the previous reg expression and also 0 (since 0 is not positive), so we add a [+-]? in the beginning which just means there is a sign in the beginning or none. So the expression is: **([+-]?[1-9][0-9]+ | 0)**

g) To be able to match all positive floating point numbers, the expression must match the decimal part in the beginning, then a literal '.' character which is the expression
? as also decimal numbers are including in floating points (with non-decimal part equal to 0), and then an arbitrary number of digits for the right side of the '.'. Since .5 is also a floating point and as stated above we need to also take care of decimals, we put a '+' in the end of the expression and only a '*' in the beginning. So we get the expression: **([0-9]*\.)?[0-9]+**
In case we also want to include floats such as 1. (for 1.0), then the expression would be:
[0-9]+(\.[0-9]*)? | \.[0-9]+

h) The first thing that we notice is that all the string that should be accepted have a 'p' and a 't' with exactly one character in between that is either a space or a letter. For the other part, any lowercase letter may be at the start and end of the string so the expression is:
[a-z]*\p[a-z][t][a-z]* Notice that for matching 't' I used [t] instead of \t as it may be mistaken for a tab.

Problem 11.3

Solution:
a) The string must start with an 'a', followed by at least 1 'b', and then end with a 'c'. The only string that matches is "abc", so the correct answer is (1).

b) The strings must start with an 'a', followed by any character but newline, and then followed by at least one of the characters 'b' or 'c'. So the strings that match are (1), (2), (3), (4), and (6).

c) The string must start with at least one "very " (so (2) does not match), followed by an optional "happy ", which can be only once or not be there at all, then followed by any of the three words "CS", "IMS", or "ECE" only once (so (1) does not match), and with student in the end. So the strings that match are (3), (4), and (5).

d) The string must start with a "<" character and end with a ">" character, and in between it must have at least one character and these characters can be anything but ">". So the strings that match are (1), (3), and (5) (Basically all single HTML tags are matched).