Register Allocation

INTRODUCTION

The focus of this project was on different register allocation heuristics for the back end of a compiler. The three heuristics are:

1. Simple Top Down Allocation
2. Live Range Top Down Allocation
3. Bottom Up Allocation

This report will begin by going over what each of these techniques include and finish by comparing the results of allocation using variable parameters in an attempt to analyze which is better to use in different cases.

HEURISTIC DESCRIPTIONS

SIMPLE TOP DOWN ALLOCATION

Simple top down allocation places the emphasis on the frequency of different registers. Taking a global frequency count for a basic block, the allocation then takes into account the number of registers available. If there is a feasible set of registers f, and k physical registers, then the allocation algorithm considers allocating $k - f$ registers. This is done simply by assigning the highest frequency registers to those registers. The assignment in this case is global and will never change. All other registers are considered spill registers. It is worth noting that in the case of a tie, the choice of register is arbitrary.

LIVE RANGE TOP DOWN ALLOCATION

Similar to simple top down allocation, here we consider feasible registers f and k physical registers. Here first the algorithm requires the additional calculation of the number of live registers per line for the entire program. Knowing which registers are live at a given point is crucial. Going line by line through the program, the algorithm looks for the first line where the number of live registers on a given line is larger than can be held by $k - f$ registers. When this happens, a register is removed from this line and deemed to be a spill register. The conditions for removal are:

1. Number of occurrences of virtual register.
2. Length of live range. (Tie breaker)

Removing registers occurs until there are enough non-feasible registers to hold the demand on each line.

After this phase is complete, each virtual register not deemed to be spilled gets mapped to a physical register. Care here is essential. Once the register is picked then that particular register must be removed as an option from all other lines. This edge case can cause problems in assignment.

Bottom Up Allocation

Bottom up allocation does not keep two sets of registers, feasible and allocable registers as before. All registers are capable of being assigned a mapping to a virtual register at any point. The algorithm starts by handing out physical registers to virtual registers until there are no more physical registers to give out. When there are no more physical registers to be issued, the fun begins!

To help with the process of choosing which register to spill to memory in order to free a physical register for the next incoming virtual register, metadata including the line number for each virtual register is used. The virtual register that gets spilled to memory is the one that is used furthest in the program. If a virtual register is never used again, then that is the obvious candidate with a distance to next use being infinity. Any time a virtual register's value is needed and it's been previously spilled, the same decision process decides which physical register it will come back to. Each virtual register is not necessarily assigned to only one physical register only in this case!

Summary of Implementation

Through the implementation of all of these algorithms no changes were made. Since I used scheme, all of the metadata was kept in a variety of different lists. The main data structure used throughout the program, however, was a structure that kept all of the state at any given point of time. This included, but is not limited to, remaining program to parse, allocated program, current line number, physical to virtual register mapping, and others. Throughout the main loop of the program, this data structure was updated accordingly each time a change was made. The final allocated program is printed out whenever there is no code to parse in the remaining program variable.

RESULTS

EXECUTION TIMES

The following tables summarizes the execution time of the written alloc code for each heuristic over:

Simple Top Down

Report	Five Registers	Ten Registers	Twenty Registers
1	0.064	0.055	0.067
2	0.045	0.041	0.046
3	0.033	0.035	0.048
4	0.046	0.040	0.042
5	0.050	0.042	0.051
6	0.041	0.036	0.050

Live Range Top Down

Report	Five Registers	Ten Registers	Twenty Registers
1	0.056	0.044	0.050
2	0.042	0.033	0.038
3	0.042	0.036	0.037
4	0.039	0.040	0.042
5	0.029	0.035	0.035
6	0.042	0.051	0.040

Bottom Up

Report	Five Registers	Ten Registers	Twenty Registers
1	0.061	0.058	0.056
2	0.042	0.047	0.054
3	0.047	0.041	0.056
4	0.041	0.044	0.046
5	0.045	0.046	0.041
6	0.040	0.044	0.042

Extremes

The smallest time is 29 milliseconds while the largest time is 67 milliseconds.

Operation Count

Here the following tables summarize the number of operations performed per allocation. Each table contains the default number of operations for comparison:

Simple Top Down

Report	Original	Five Registers	Ten Registers	Twenty Registers
1	62	192	172	140
2	50	141	126	96
3	49	122	102	70
4	36	104	89	65
5	44	133	118	89
6	47	142	127	98

Live Range Top Down

Report	Original	Five Registers	Ten Registers	Twenty Registers
1	62	117	90	63
2	50	67	51	51
3	49	97	77	50
4	36	52	37	37
5	44	131	104	69
6	47	72	48	48

Bottom Up

Report	Original	Five Registers	Ten Registers	Twenty Registers
1	62	144	119	99
2	50	83	78	68
3	49	88	73	57
4	36	64	59	49
5	44	NA	NA	65
6	47	86	81	76

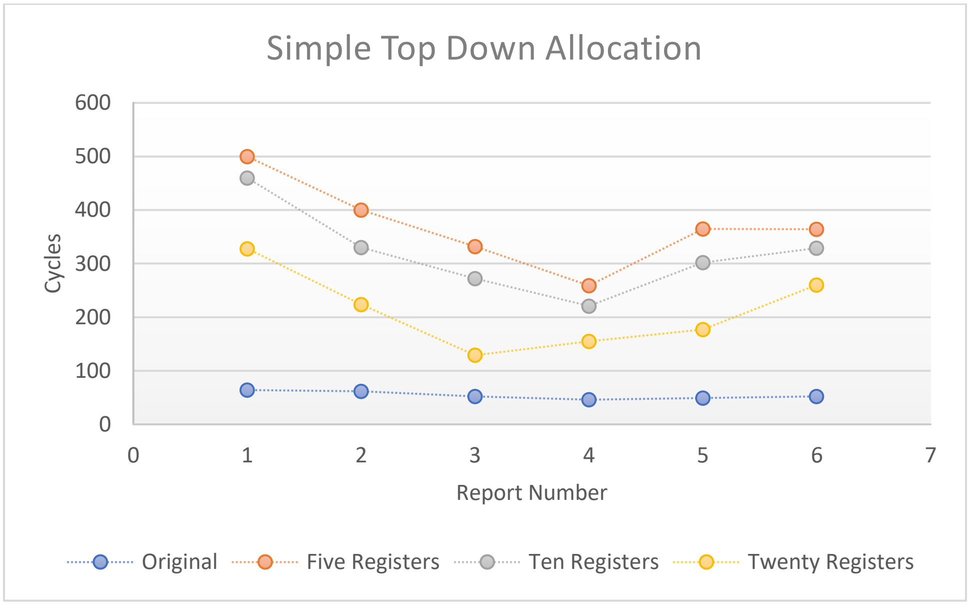
Notes

The NA values in the bottom up allocation table have to do with a bug that was unfixed before submission. For these two runs, the program does not successfully perform allocation.

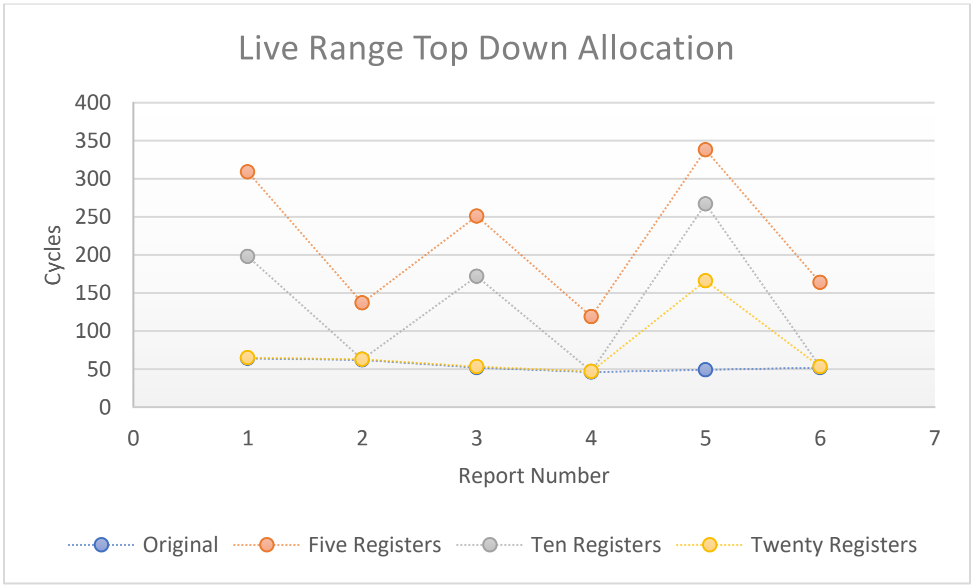
CYCLES

For this metric, the data is shown via scatter plots instead of in tabular form. It is also presented in multiple ways to aid in drawing conclusions:

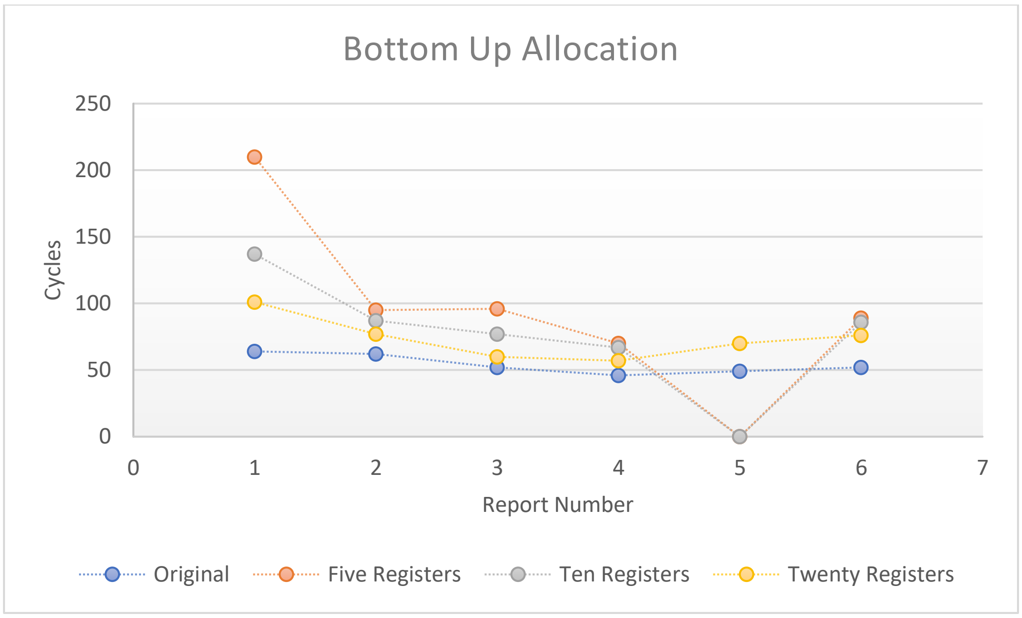
Simple Top Down



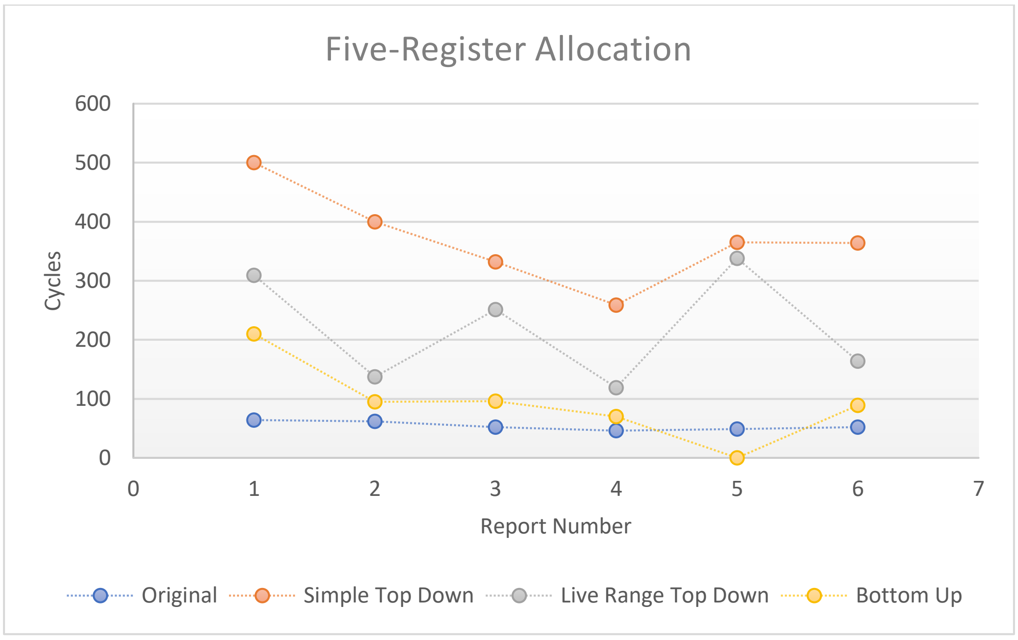
Live Range Top Down



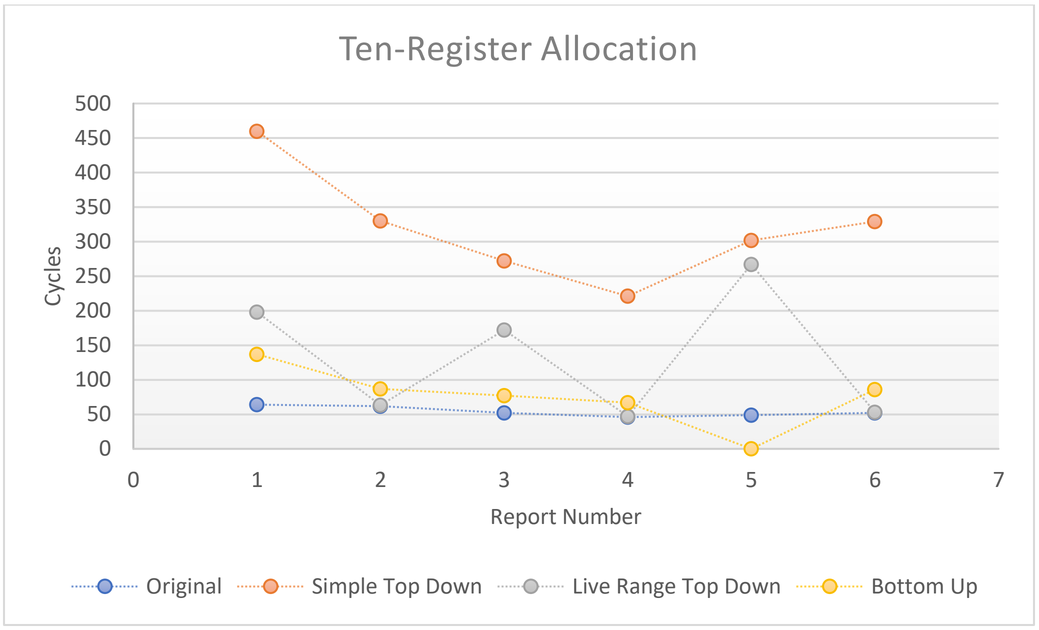
Bottom Up



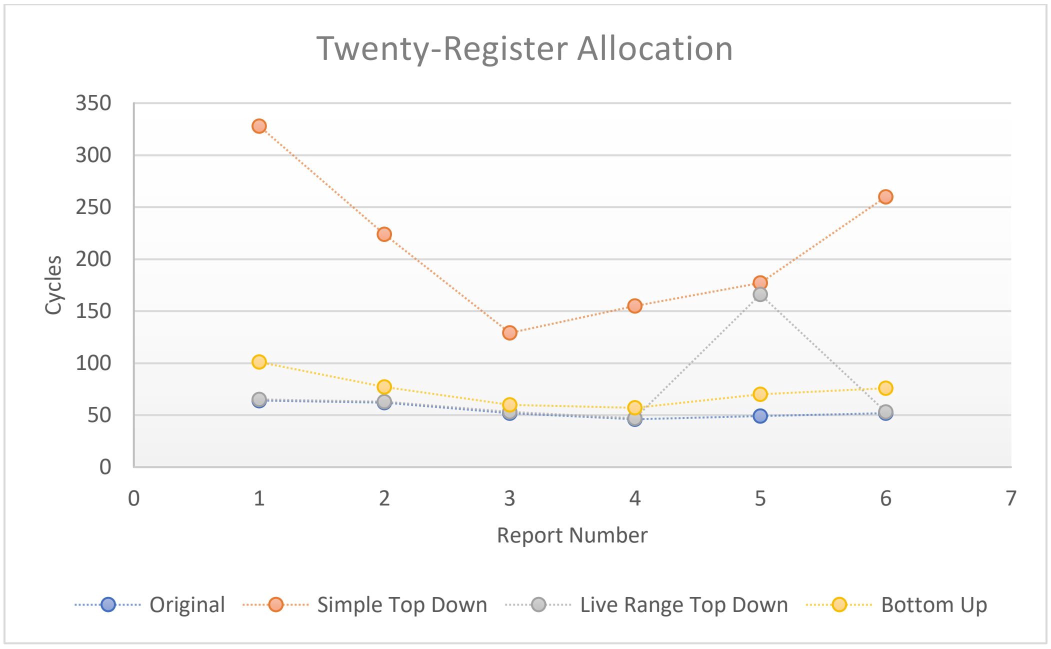
Five-Register Allocation



Ten-Register Allocation



Twenty-Register Allocation



Analysis

Time

For all times, the order of magnitude is the same. Since the difference between each allocator is not significantly different within the time metric, there is no disadvantage to choosing one or the other here.

Operation Count

Here we notice a couple of trends:

- Operations decrease as number of hardware registers increase.
- As number of hardware registers increase, convergence to optimal operations is fastest in live range top down allocation.

Outside of these observations, since operation count and cycles are tied together by their nature, we move on to cycles.

Cycles

Looking at the individual plots for each heuristic, a general trend that follows in all cases is that the number of cycles goes down for each additional register the algorithm has access to. Of the three, bottom up allocation starts with the lowest cycle count, but it overtaken by the quick convergence of live range top down.

Using the three plots on five-register, ten-register, and twenty-register machines we see that in all cases, simple top down produces the worst cycles. Looking at the grey line representing live range top down, we see that it's all over the place for cycle counts at five registers but overtakes two of bottom up's cycle counts at ten registers. Live range top down takes over five of bottom ups at twenty registers. Again, we notice fast convergence for live range top down.

Conclusion

We can use the observations in the previous section to help decide when we would want to use a particular heuristic for register allocation. In the case of simple top down allocation, the result is never. It performs worst in all situations tested.

The case is not cut and dry for bottom up and live range top down, however. In these cases, it appears that if you're on a low register machine, the initial low count of cycles for bottom up allocation is a clear choice. As you get to machines with more registers, however, the better choice is live range top down allocation. It converges very quickly to the number of cycles the program uses before allocation even begins.