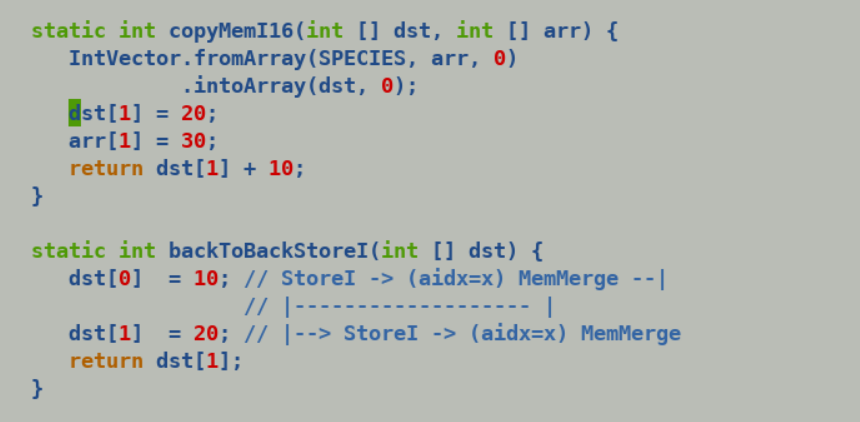
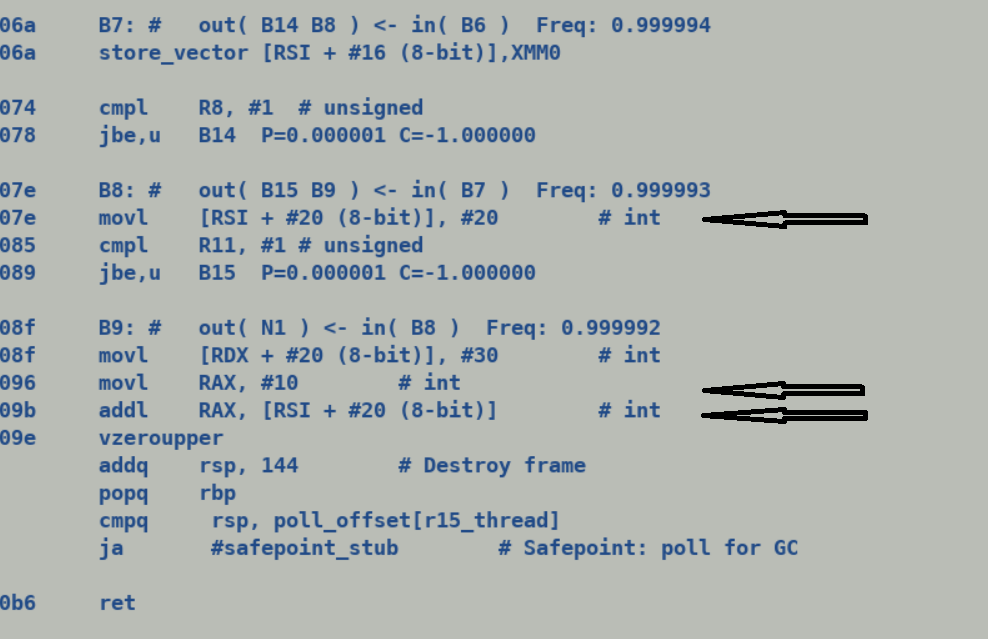
Memory Aliasing Implimentation

Memory Alias ( store -> MergeMem (idx=X) -> store -> MergeMem (idx=X) -> Load)

*find\_previous\_store* accesses a store with a different base at alias idx = X thus preventing value forwarding.





This could be improved by generating a new MeregMem node with each modification to memory instead of updating alias index of MergeMem node at SafePoints, each merge memory node will still add a new memory modifying inputs at particular alias index e.g

Memory (state of memory before store)

Dst[1] = 20; -> StoreI Mem , Dst+4, 20

MergeMem (state of memory before above store) + new storeI input at alias index corresponding to TypeAryInt\*

Arr[1] = 10;

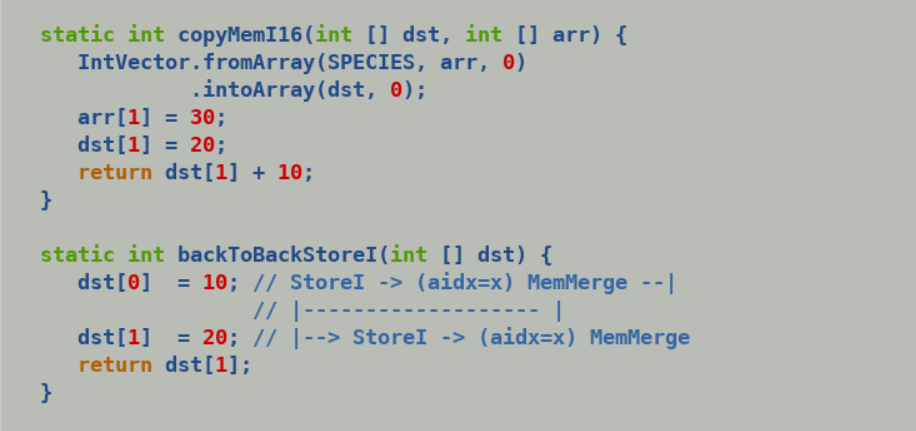
MergeMem(state of memory before above store) + new storeI input at alias index corresponding to TypeArrInt\*)

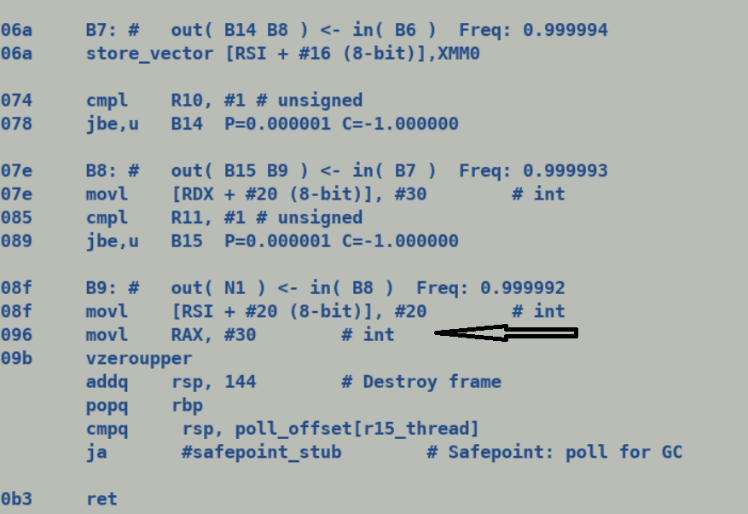
return dst[1] + 10;

But, given that java arrays are heap allocated and it may happen that multiple array arguments of callee are passed same actual array. Thus in such a case both Dst[1] and Arr[1] are aliases and fetching value from dst[1] will be incorrect forwarding

*“Memory merge nodes act as book keeping node to represent the state of memory, its used to quickly extract the input feeding to an aliased memory. Its main utility is during IR construction phase, I assume you encountered an assertion which expects a specific graph shape but introducing an empty memory merge node to bypass assertion can also be replaced by extending the assertion itself to have phase check which prevent assertion failure in any non-IR construction pass (parsing) pass.”*

*find\_previous\_store* encountering store with same alias index can forward the initialing values of store to its consumer, which is constant folded and moved to the register.





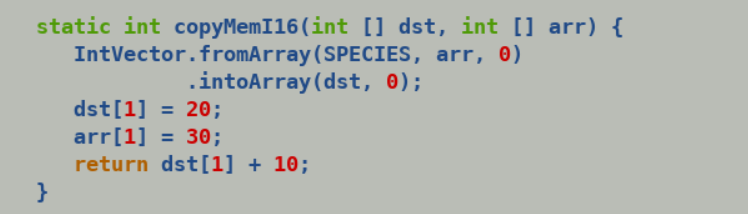
Store VAL , X

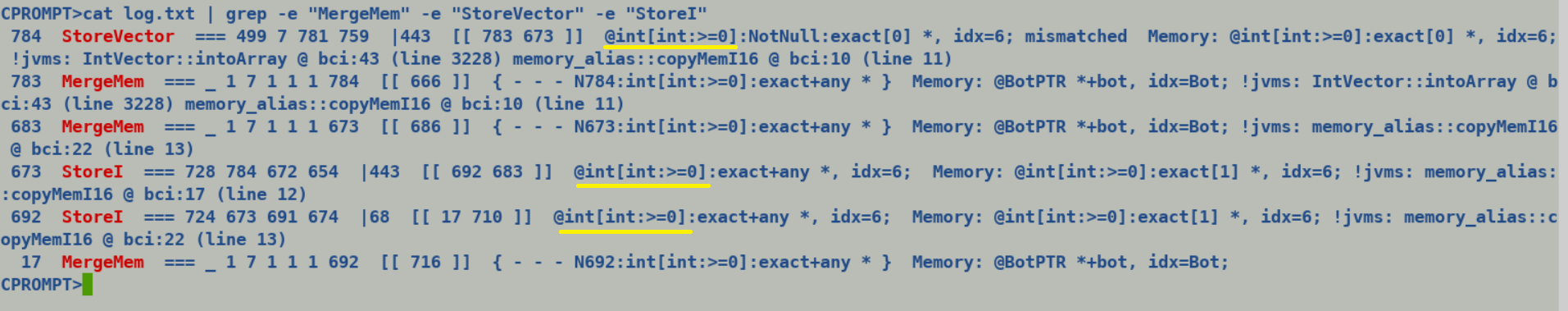
MemoryMerge AIDX = X

T1 = Load X

T1 += 10;

Store vector (with array as backing storage [I ) aliases to subsequent scalar store to same memory since both of them access memory with same alias\_type (TypeArrPtr [I) , this will automatically result into fetching preceding StoreVector as the incoming memory edge to subsequent StoreI and will enable scheduler to constrain their ordering.





Existing flow:

VBox -> NewArray -> StoreVector VAL , NewArray + BoxAllocation + Putfield

-----------------

With Multi-field:

VBox -> BufferAllcate -> StoreVector VAL, Buffer + BoxAllocate + PutField

Alias type of StoreVector to backing storage buffer should be similar to alias type of store vector to backing storage array , to add strict ordering and store bypassing constraints (although buffer elements are initialized by) var handle unsafe APIs.